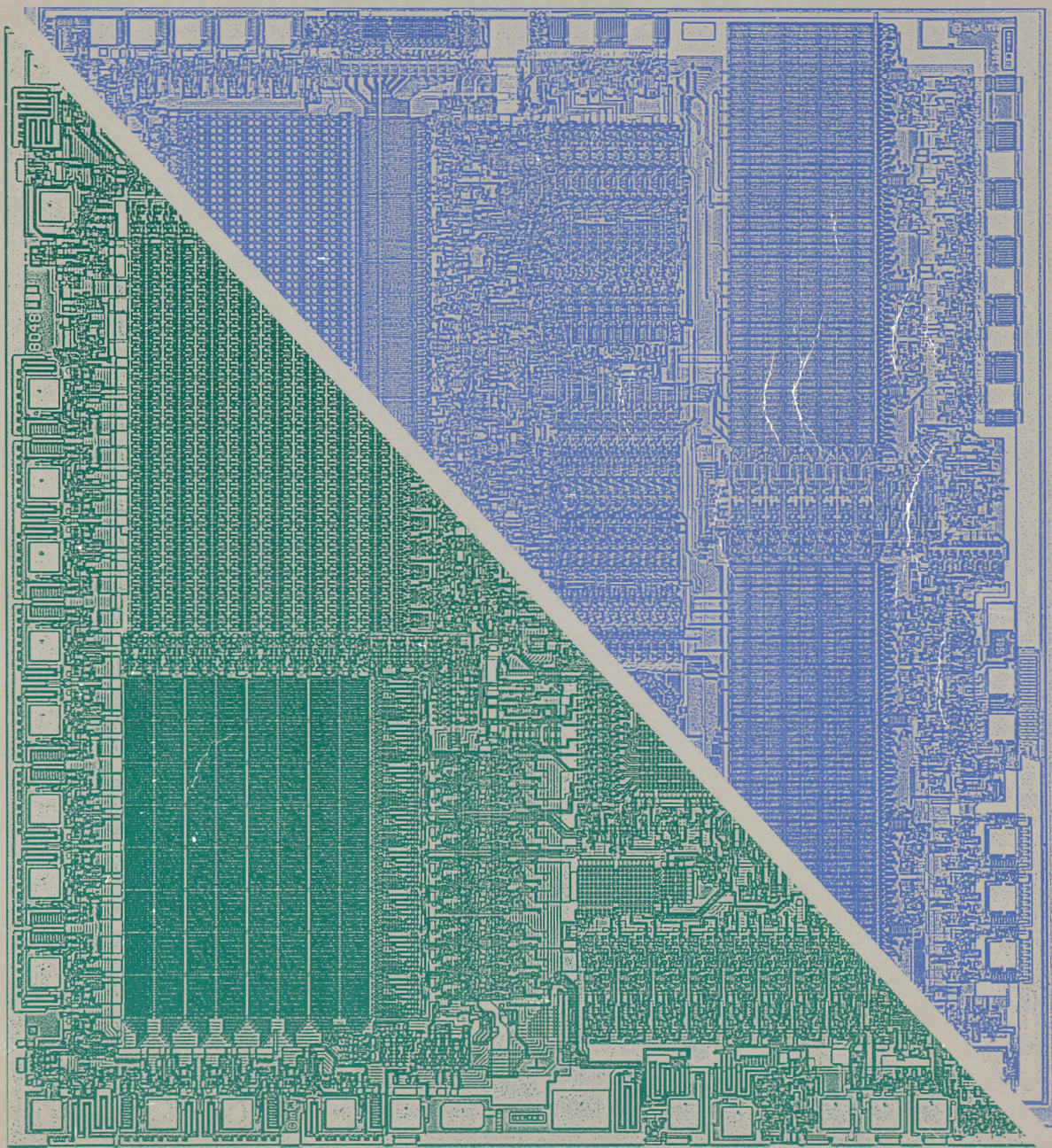


intel

Microcontroller Applications Handbook



Order Number 210316-001

intel®

MICROCONTROLLER APPLICATIONS HANDBOOK

FEBRUARY 1982

FEBRUARY 1982



MICROCONTROLLER APPLICATIONS HANDBOOK

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

BCP, CREDIT, i, ICE, iCS, i_m, iMMX, Insite, Intel, int_el, Intelevison,
Inteltec, iOSP, iRMX, iSBC, iSBX, Library Manager, MCS,
Megachassis, Micromainframe, Micromap, Multimodule,
Plug-A-Bubble, PROMPT, RMX/80, System 2000 and UPI.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

* MULTIBUS is a patented Intel bus.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Department SV3-3
3065 Bowers Avenue
Santa Clara, CA 95051

Table of Contents

CHAPTER 1	
Introduction	1-1
CHAPTER 2	
MCS [®] -51 Family	2-1
MCS [®] -51	
A Microcomputer Optimised for Control	2-1
Microcontroller Doubles as Boolean Processor	2-7
An Introduction to the Intel MCS-51 Single-Chip Microcomputer	2-13
Using the Intel MCS-51 Boolean Processing Capabilities	2-43
CHAPTER 3	
MCS [®] -48 Family	3-1
MCS [®] -48	
Single-Chip 8-Bit Microcomputer Fills Gap between Calculator	
Types and Powerful Multichip Processors	3-1
Application Techniques for the MCS-48 Family	3-9
Keyboard/Display Scanning with Intel's MCS-48 Microcomputers	3-33
Serial I/O and Math Utilities for the 8049 Microcomputer	3-59
A High-Speed Emulator for Intel MCS-48 Microcomputers	3-83
Using the 8049 as an 80 Column Printer Controller	3-187
Microcontroller Includes A-D Converter for	
Lowest-Cost Analog Interfacing	3-217
Microcomputer's On-Chip Functions Ease Users' Programming Chores	3-223
Designing with Intel's 8022 Microcomputer	3-229
Energy Savings in an Induction Motor Using the 8022 Microcontroller	3-247
CHAPTER 4	
General	4-1
Application of Intel's 5V EPROM and ROM Family for	
Microprocessor Systems	4-1
CRYSTALS: Specifications for Intel Components	4-9

This manual is a collection of the application information available for the MCS-48® Family and the MCS-51® Family of single-chip microcontrollers. The MCS-48 family has become an industry standard since the introduction of its original member, the 8748, in 1977. The family now consists of eleven members (see Table 1). All of these components share a common architecture, and each of them has unique features which might prove beneficial in a given application. The MCS-51 family — introduced in 1980 — features significant performance and architectural improvements over existing microcontrollers. Currently,

the family consists of three members: the ROMless 8031, the 8051, and the EPROM-based 8751. The high performance of the MCS-51 family opens up a new spectrum of sophisticated designs for microcontrollers. The material included in this manual is believed to be accurate; however, some of the crystal circuits are outdated. The recommended crystal circuits can be found in the User's Manual. If you find any additional errors, or if you have suggestions for future application notes for the MCS-48 or the MCS-51 families, we would appreciate hearing from you.

Table 1. Characteristics

Component	ROM Size (kilobytes)	RAM Size (Bytes)	I/O Pins	Cycle Time (micro sec.)	AD Channels	Timer/ Counters
8020H	1	64	13	8.5	—	1-8 bit
8021H	1	64	21	8.5	—	1-8 bit
8022	2	64	28	8.5	2	1-8 bit
8035H	—	64	15	1.9	—	1-8 bit
8039H	—	128	15	1.4	—	1-8 bit
8040	—	256	15	1.4	—	1-8 bit
8048H	1	64	27	1.9	—	1-8 bit
8049H	2	128	27	1.4	—	1-8 bit
8050	4	256	27	1.4	—	1-8 bit
8748	1*	64	27	2.5	—	1-8 bit
8749H	2*	128	27	1.4	—	1-8 bit
8031	—	128	16	1.0	—	2-16 bit
8051	4	128	32	1.0	—	2-16 bit
8751	4*	128	32	1.0	—	2-16 bit

*Erasable EPROM

2

MCS-51 Family

ELECTRONIC PRODUCT DESIGN

MCS-51: a microcomputer optimised for control

A microcomputer optimised for control applications needs fast, real-time operation, rapid context switching facilities and efficient bit and byte operations, writes Intel's Howard Kornstein who describes how the MCS-51 family fits these requirements.

Two unique structures form the foundation of microelectronic computers—the microprocessor and the microcomputer. Microprocessors utilise most of a silicon die for central processing unit resource. Here, the driving philosophy of chip design is to maximise the CPU architecture to serve applications problems requiring high performance processing. The microcomputer, on the other hand, utilises the available silicon die to provide a total computer structure. This will encompass the central processing unit as well as program memory, data memory, input/output, interrupt unit and ancillary functions. The emergence of an entire computer structure on a chip was a significant advance in semiconductor technology—certainly as significant an advance as the creation of the microprocessor itself.

With the introduction of the single

chip microcomputer, the application range available for computerisation was increased in a highly significant way. The most popular applications of microcomputers was in electronic logic replacement or electro-mechanical control system replacement. Typical examples of such applications are replacement of electro-mechanical washing machine controller elements or discrete logic traffic light control systems.

Single chip microcomputers were also very successful in applications involving control of peripherals, as a slave computer for a minicomputer or a master microcomputer. Sequencing systems of different types were a common implementation for the single chip computer, and so were dedicated closed loop systems. Microcomputers have also been used to produce energy systems for heating control in both industrial applications

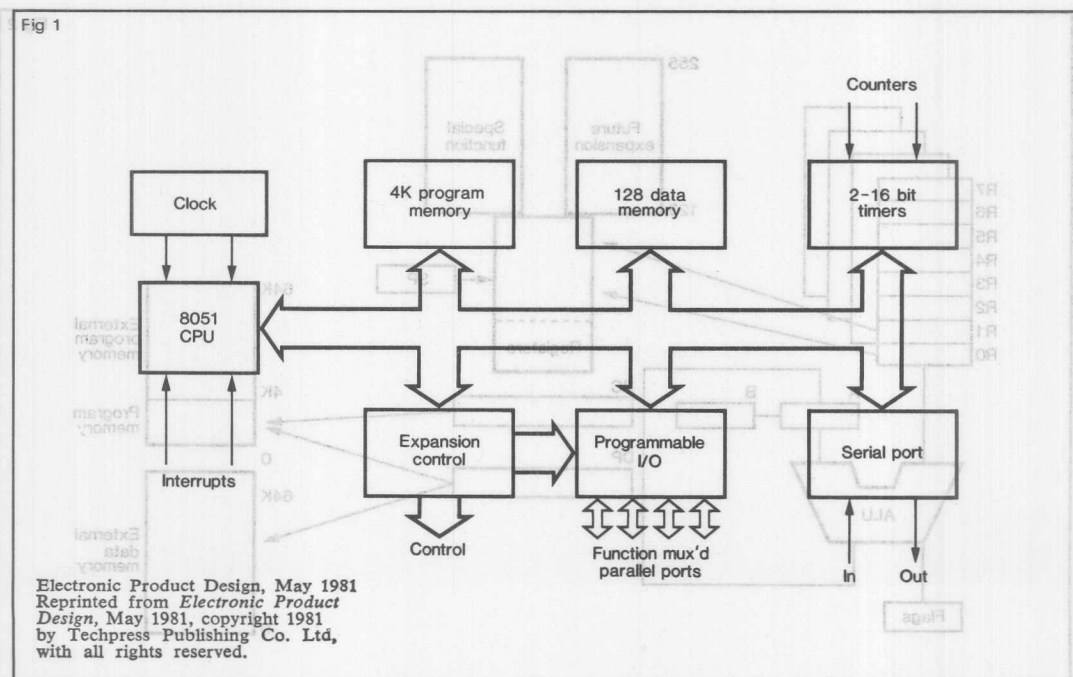
and in conventional central heating systems.

Looking at these diverse applications of single chip microcomputers we can see that "typical" microcomputer applications represents a control-orientated type of activity—where the microcomputer is involved in a closed loop system, operating in real-time, dedicated toward some specific type of control function. Because of this strong orientation towards control, successful microcomputer families have been structured with an architecture that is well suited for dealing with the control environment. Control in general is typified by real-time processing, high speed execution, fast context change and facilities for effective decision making and logic implementation.

The first such microcomputer family was Intel's MCS-48 range. Because this family of devices was particularly orientated towards control applications it was dubbed a "micro-controller" class of computer. The central member of the MCS-48 family was the 8048 which provided one thousand words of pro-

Fig. 1: 8051 block diagram

Fig 1



gram memory, 64 bytes of read/write data memory, a comprehensive instruction set that was control oriented, an 8 bit-central processing unit, 27 input/output lines, a real time clock, clock generator and interrupt unit all on a single chip. This device became the industry's standard microcomputer.

Many other family members soon followed; these included the 8021, a very low cost version of the 8048. Another device, the 8022, not only has an 8048 structure on chip but also a complete analogue sub-system including A/D converter, analogue multiplexer and signal conditioning ports. The 8041 is a slave microcomputer for hierarchical systems; the 8049 is a high performance 8048 with twice the program memory and data memory area as the 8048. Establishing a family of devices with a common architecture provided to systems designers the ability to pick a processor particularly cost effective in their application.

It is worthwhile to look at the facilities necessary in a microcomputer and see how the 8051, Intel's fourth generation microcomputer, provides these facilities. The most critical requirement of a microcom-

puter's design is to maximise the effective use of the on-chip resources. After all, a single chip microcomputer has a fundamentally finite resource.

For this reason, every resource on the chip must be used to maximum effect. This is particularly important in the instruction set, where one has to consider the fixed amount of program memory space on the chip. There is no room for inefficiency in code compaction for a device that has such finite resources.

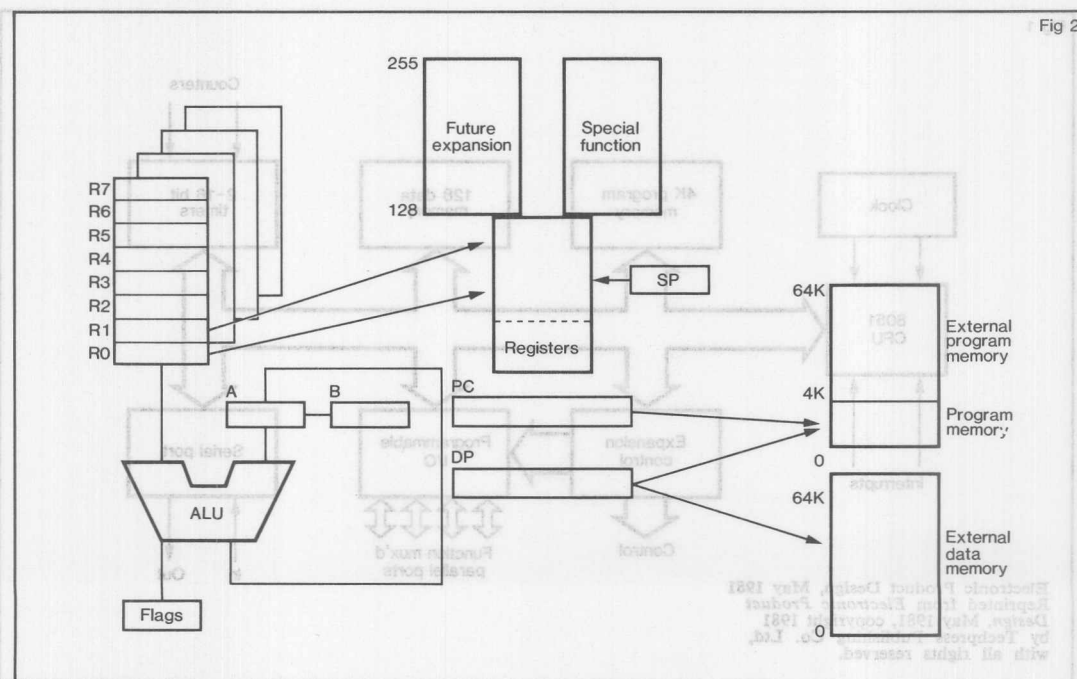
The 8051 provides several approaches for maximising the efficiency of this memory resource. The simplest way that the 8051 addresses the finite resource problem is to quadruple the amount of program memory space provided above the basic 8048 (4k of program memory on chip). The instruction set of the 8048, which has always been efficient in areas of code compaction and compaction efficiency, has been improved in the 8051. Over 50 per cent of the instructions of the 8051 are single byte, and over 80 per cent are no more than two bytes. Many "efficient" new instructions have been added to the 8051; for example, memory to memory moves, compare

and branch if not equal, and multiply and divide.

By not employing a conventional Von Neuman structure (common data and program space), very compact addressing forms are possible for on-chip data memory. The input/output resources of the 8051 are memory mapped and comprise four input/output ports totalling 32 I/O bits in a 40 pin package. The I/O can be configured by a "quasi bi-directional" port which allows any bit on the port to be used either as input or output, and allows for read-modify-write operations.

To match the control orientation of most single chip applications the 8051 provides specific facilities that enhance decision making. The 8051 provides a "jump-on-comparison" within a decision table (the "do case" statement), a very fast multiply and divide (4 microseconds) which facilitates high speed servocontrol computations, enhanced tale lock-up facilities and a very sophisticated instruction subset dealing with single bit variables. So comprehensive are the 8051 bit-orientated instructions that this group of instructions,

Fig. 2: 8051 registers



associated bit memory and I/O provides a "processor within a processor"—the Boolean processor.

The Boolean processor consists of a one bit accumulator, the carry flag, and 128 directly addressable bit memory variables in the data memory. Another 128 bit variables are specified in the input/output or functional control address space. The Boolean operators include clear, set, complement, and, or, move and jump-on bit. With the Boolean processor, it is possible to directly convert complex combinational or sequential logic equations into a program implementation executed at high speeds. An example of such an implementation is shown in Fig. 3.

One of the facilities that allows high speed control in the 8051 is its advanced interrupt structure. The interrupt unit of the 8051 provides five sources of interrupt, two of which are from external conditions, two from on-chip counter timers and one from the serial I/O port. The interrupt logic provides two levels of interrupt priority, as well as having individual interrupt masking and global interrupt enable or disable.

To provide very fast context switching in the 8051, four on-chip register banks are incorporated in the device. This eliminates the need for saving registers on the occurrence of an interrupt. The device also contains a conventional stack and stack pointer register. Perhaps what makes the 8051 so effective in control is the basic execution cycle time—most instructions for the 8051 are executed in one microsecond. The device is implemented in Intel's HMOS high-speed technology.

An important aspect of microcomputer implementation is the provision of portability of software over a range of applications. As was mentioned, the MCS-48 family includes many devices, the 8021, 8022, 8048, 8041, 8049, all aimed at providing a suitable architecture to match a differing range of cost and performance criteria in varying microcontroller applications. These computers use a common architecture, and a common instruction subset runs through all of the devices, although the 8048 and 8049 provide a more enhanced instruction set. The 8051 provides an instruction set for very high performance single chip microcomputer applications but still provides code com-

BFUNC3 Solve a random logic function of 6 variables using straight-line logical instructions on MCS-51 Boolean variables.

```
MOV C,V
ORL C,W ; Output of OR gate
ANL C,U ; Output of top AND gate
MOV FO,C ; Save intermediate state
MOV C,X
ANL C,Y ; Output of bottom AND
gate
ORL C,FO ; Include value saved above
ORL C,Z ; Include last input variable
MOV G,C ; Output computed result
```

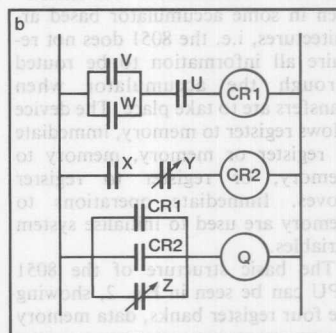
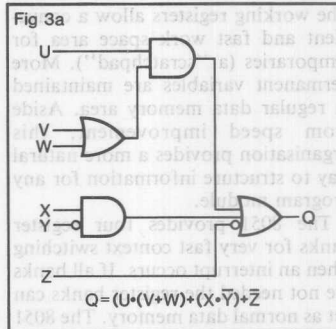
Fig. 3: TTL (a) and relay logic (b) implementations of a Boolean function. The table above is a software version of the same function using the 8051's direct bit addressing to achieve maximum coding efficiency

patibility with its predecessors in the MCS-48 range. A subset of the 8051 instruction set covers the entire range of Intel's MCS-48 line from the 8021 through to the 8049 device. This allows for portability of 8051 code upward from these 802X and 804X devices and portability downward from the 8051 into these other members of the Intel single chip family.

It is important that single chip microcomputers have expansion capabilities beyond the single chip level, otherwise a single chip system can be a potential bottleneck to configuration flexibility. If any of the on-chip resources must be exceeded, whether it be program memory, data memory, I/O, interrupt facility, number of counter timers and if no expansion path is provided beyond the single chip itself, a designer can run into a catastrophic constraint. The single chip microcomputer must be easily expanded in any of these resource areas.

The 8051 is expandable in all of these directions. The code space and data memory space can be extended to 128k. Program memory of data memory external to the 8051 can be standard ROM, EPROM or RAM memory, or can be highly integrated members of the 8085 family. These provide not only memory expansion but also I/O expansion with every chip added.

Furthermore, the 8051 provides expandability through networking. This is in keeping with recent trends in providing distributed computing in microcomputer systems. The 8051



provides a full duplex serial communications port, on-chip. One of the ways that this port can be utilised is in a multi-processor configuration, using a protocol in which one of the bits in the serial word is defined as an "attention bit" which notifies 8051s in the network to look at a received control word to determine if it is being addressed to perform a function. To facilitate ease of design, there is a pin compatible EPROM version of the 8051 designated the 8751. This allows a design to progress from prototyping into production without any complications in testing and also allows trial-marketing of an 8051 based product.

Architecture

The 8051 microcomputer is a register orientated architecture. The device has an accumulator and eight working registers, two of which are indirect pointers into the data memory area. The registers are designated R0 to R7. The organisation is in keeping with the philosophy that a register orientated machine is particularly effective in both control and general purpose computing applications by providing an effective way of separating the temporaries and variables in a system program.

The working registers allow a convenient and fast work space area for temporaries (a "scratchpad"). More permanent variables are maintained in regular data memory area. Aside from speed improvement, this organisation provides a more natural way to structure information for any program module.

The 8051 provides four register banks for very fast context switching when an interrupt occurs. If all banks are not needed the register banks can act as normal data memory. The 8051 has no "accumulator bottleneck" seen in some accumulator based architectures, i.e. the 8051 does not require all information to be routed through the accumulator when transfers are to take place. The device allows register to memory, immediate to register or memory, memory to memory, or register to register moves. Immediate operations to memory are used to initialise system variables.

The basic structure of the 8051 CPU can be seen in Fig. 2, showing the four register banks, data memory area, program memory area, arithmetic logic unit and the accumulator. Register B is used as an extension register during the operations of multiply and divide. The program counter of the 8051 is 16 bits long, thus providing facility of addressing 64k of program memory. The 8051 automatically recognises the address space which is on-chip and that which is off-chip. A program reference into address 4096 or greater is automatically generated off-chip. A member of the MCS-51 family, the 8031, has all program memory off-chip. The 8031 is a low-cost member of the 8051 family and provides another way for low production runs to be realised in EPROM based memory.

A second pointer register in the 8051 is the data pointer. This is a 16-bit register pointing into either program or data memory off-chip. The data pointer provides a base register for table look-up in ROM, external data transfers, or into jump tables which are held in the program memory area. The data pointer may be manipulated as a 16-bit number or as a separate data pointer high/data pointer low locations. The data pointer may initialise with a 16-bit immediate move and it can be indexed.

Two other memory address registers have been mentioned, R0 and R1. These registers can be utilised as pointers for indirect moves in the on-chip data memory area. An 8-bit stack pointer register also points into the RAM area. The 8051 thus provides conventional push and pop operations, subroutine calls and returns which can be nested, and the ability to save program counter and status information on the occurrence of interrupt.

The data memory area of the 8051 is separated into normal data memory and an area known as special function registers. The special function registers provide a memory mapped register facility for controlling resources and accessing input/output. Registers in the special functions area include the accumulator, B register, program status word, interrupt unit control registers, timer-counter registers and the input/output registers of the 8051.

Memory mapped I/O is a very useful capability in I/O handling. It allows instructions such as compare, logical instructions or arithmetic operations to take place between input/output ports and the accumulator, or immediate operations to manipulate I/O.

Instruction set

The 8051 provides an instruction set which is a superset of the MCS-48

family, which is highly compact and symmetrical. Instruction classes for the 8051 include arithmetic operations, logical operations, data transfers, Boolean variable manipulation and program and machine control instructions. The arithmetic operations include the add, subtract, multiply, divide, increment and decrement. Logical operations include and, or, exclusive or, complement as well as the rotate instructions. Data transfers include move operations, push and pop and the exchange instruction. Boolean variables allow the programmer to set or clear a bit, complement a bit and perform and, or and move bit.

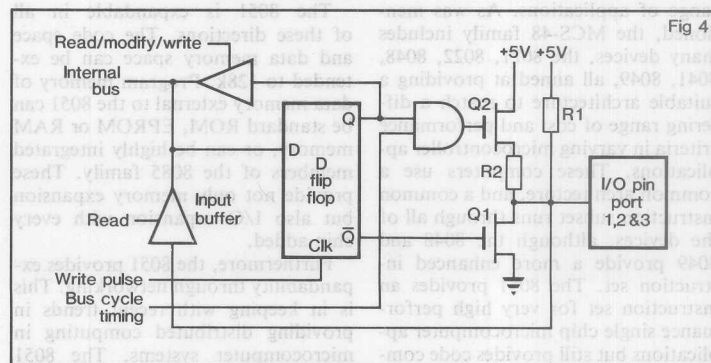
Program control instructions include jumps, calls, special jump instructions that allow jump tables or short "looping" jumps. The 8051 has four basic addressing modes, register, direct, register indirect, and immediate. The device also provides comprehensive addressing dealing with tables. This is particularly important in microcontroller-type applications.

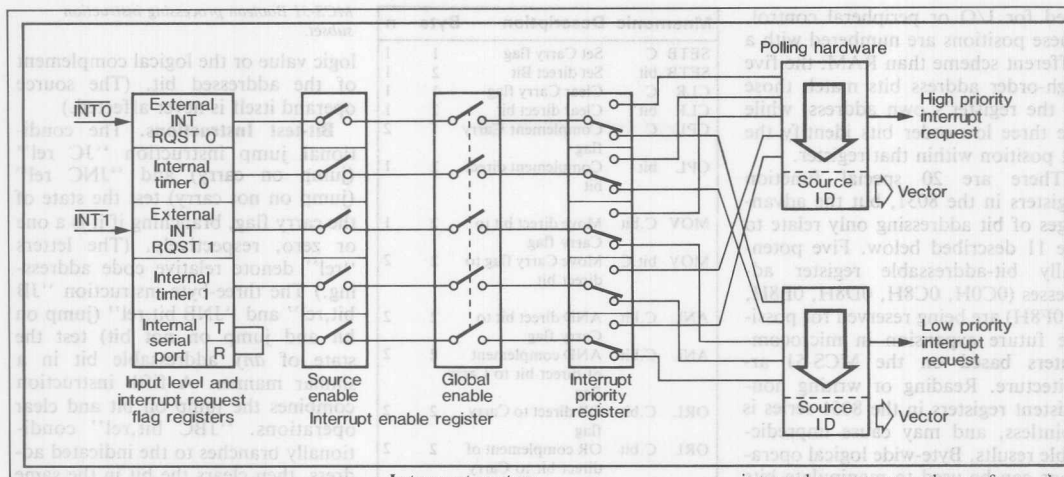
Bit operations

One of the factors determining how long it will take a microcomputer to complete a given chore is the number of instructions it must execute. What makes a given computer architecture particularly well—or poorly—suited

Fig. 4: The I/O port versatility results from the "quasi-bidirectional" output structure depicted below. An output latch bit associated with each pin is updated by direct addressing instructions when that port is the destination. The latch state is buffered to the outside world by R1 and Q1, which may drive a standard TTL input. (In TTL terms, Q1 and R1 resemble an open-collector output with a pull-up resistor to Vcc).

Operations using an input port or pin as the source open and use the logic level of the pin itself, rather than the output latch contents. This level is affected by both the microcomputer itself and whatever device the pin is connected to externally. The value read is essentially the "OR-tied" function of Q1 and the external device. If the external device is high-impedence, such as a logic gate input or a three state output in the third state, then reading a pin will reflect the logic level previously output.





for a class of problems is how well its instruction set matches the tasks to be performed. The better the 'primitive' operations correspond to the steps taken by the control algorithm, the lower the number of instructions needed, and the quicker the program will run. All else being equal, a CPU supporting 64-bit arithmetic directly could clearly perform floating-point maths faster than a machine bogged-down by multiple-precision subroutines. In the same way, direct support for bit manipulation naturally leads to more efficient programs handling the binary input and output conditions inherent in digital control problems.

Let's see how the four basic elements of a digital computer—a CPU with associated registers, program memory, addressable data RAM, and I/O capability—relate to Boolean variables.

CPU. The 8051 CPU incorporates special logic devoted to executing several bit-wide operations. All told, there are 17 such instructions, all listed in the table. Not shown are 94 other (mostly byte-oriented) 8051 instructions.

Program Memory. Bit-processing instructions are fetched from the same program memory as other arithmetic and logical operations. Several sophisticated program control features like multiple addressing modes, subroutine nesting, and a two-level interrupt structure are useful in structuring Boolean processor-based programs.

Boolean instructions are one, two,

Interrupt system

External events and the real-time-driven on-chip peripherals require service by the CPU asynchronous to the execution of any particular section of code. To tie the asynchronous activities of these functions to normal program execution, a sophisticated multiple-source, two-priority-level, nested interrupt system is provided. Interrupt response latency ranges from 3μs to 7μs when using a 12 MHz crystal. The 8051 acknowledges interrupt requests from five sources: Two from external sources via the INT0 and INT1 pins, one from each of the two

internal counters and one from the serial I/O port. Each interrupt vectors to a separate location in program memory for its service program. Each of the five sources can be assigned to either of two priority levels and can be independently enabled and disabled. Additionally all enabled sources can be globally disabled or enabled. Each external interrupt is programmable as either level- or transition-activated and is active-low to allow the "wire-or-ing" of several interrupt sources to the input pin. The interrupt system is shown diagrammatically above.

or three bytes long, depending on what function they perform. Those involving only the carry flag have either a single-byte opcode or an opcode followed by a conditional-branch destination byte. The more general instructions add a "direct address" byte after the opcode to specify the bit affected, yielding two or three byte encodings. Though this format allows potentially 256 directly addressable bit locations, not all of them are implemented in the 8051 family.

Data Memory. The bit instructions can operate directly upon 144 general purpose bits forming the Boolean processor "RAM." These bits can be used as software flags or to store program variables. Two operand instructions use the CPU's carry flag ("C") as a special one-bit register; in a sense, the carry is a "Boolean accumulator" for logical operations and data transfers.

Input/Output. All 32 I/O pins can

be addressed as individual inputs, outputs, or both, in any combination. Any pin can be a control strobe output, status (test) input, or serial I/O link implemented via software. An additional 33 individually addressable bits reconfigure, control, and monitor the status of the CPU and all on-chip peripheral functions (timer/counters, serial port modes, interrupt logic, and so forth).

Direct bit addressing

The most significant bit of the direct address byte selects one of two groups of bits. Values between 0 and 127 (00H and 7FH) define bits in a block of 32 bytes of on-chip RAM, between RAM addresses 20H and 2FH. They are numbered consecutively from the lowest-order byte's lowest-order bit through the highest-order byte's highest-order bit.

Bit addresses between 128 and 255 (80H and 0FFH) correspond to bits in a number of special registers, mostly

used for I/O or peripheral control. These positions are numbered with a different scheme than RAM: the five high-order address bits match those of the register's own address, while the three low-order bits identify the bit position within that register.

There are 20 special function registers in the 8051, but the advantages of bit addressing only relate to the 11 described below. Five potentially bit-addressable register addresses (0C0H, 0C8H, 0D8H, 0E8H, & 0F8H) are being reserved for possible future expansion in microcomputers based on the MCS-51 architecture. Reading or writing non-existent registers in the 8051 series is pointless, and may cause unpredictable results. Byte-wide logical operations can be used to manipulate bits in all *non-bit* addressable registers and RAM.

The accumulator and B registers (A and B) are normally involved in byte-wide arithmetic, but their individual bits can also be used as 16 general software flags. Added with the 128 flags in RAM, this gives 144 general purpose variables for bit-intensive programs. The program status word (PSW) is a collection of flags and machine status bits including the carry flag itself. Byte operations acting on the PSW can therefore affect the carry.

Having looked at the bit variables available to the Boolean Processor, we will now look at the four classes of instructions that manipulate these bits.

State Control. Addressable bits or flags may be set, cleared, or logically complemented in one instruction cycle with the two-byte instructions SETB, CLR, and CPL. (The "B" affixed to SETB distinguishes it from the assembler "SET" directive used for symbol definition.) SETB and CLR are analogous to loading a bit with a constant: 1 or 0. Single byte versions perform the same three operations on the carry.

The MCS-51 assembly language specifies a bit address in any of three ways:

- by a number or expression corresponding to the direct bit address (0-255).
- by the name or address of the register containing the bit, the *dot operator* symbol (a period: "."), and the bit's position in the register (7-0).
- in the case of control and status

Mnemonic	Description	Byte	n
SETB C	Set Carry flag	1	1
SETB bit	Set direct Bit	2	1
CLR C	Clear Carry flag	1	1
CLR bit	Clear direct bit	2	1
CPL C	Complement Carry flag	1	2
CPL bit	Complement direct bit	2	1
MOV C,bit	Move direct bit to Carry flag	2	1
MOV bit,C	Move Carry flag to direct bit	2	2
ANL C,bit	AND direct bit to Carry flag	2	2
ANL C,bit	AND complement of direct bit to Carry flag	2	2
ORL C,bit	OR direct to Carry flag	2	2
ORL C,bit	OR complement of direct bit to Carry flag	2	2
JC rel	Jump if Carry is flag is set	2	2
JNC rel	Jump if No Carry flag	2	2
JB bit,rel	Jump if direct Bit set	3	2
JNB bit,rel	Jump if direct Bit Not set	3	2
JBC bit,rel	Jump if direct Bit is set & Clear bit	3	2

n = no. of execution cycles

Address mode abbreviations:

C — Carry flag.
bit — 128 software flags, any I/O pin, control or status bit.
rel — All conditional jumps include an 8-bit offset byte. Range is +127/-128 bytes relative to first byte of the following instruction.

registers, by the predefined assembler symbols.

Data Transfers: The two-byte MOV instructions can transport any addressable bit to the carry in one cycle, or copy the carry to the bit in two cycles. A bit can be moved between two arbitrary locations via the carry by combining the two instructions. (If necessary, push and pop the PSW to preserve the previous contents of the carry.) These instructions can replace the multi-instruction sequences appearing in controller applications whenever flags or outputs are conditionally switched on or off.

Logical Operations. Four instructions perform the logical AND and logical-OR operations between the carry and another bit, and leave the results in the carry. The instruction mnemonics are ANL and ORL; the absence or presence of a slash mark ("/") before the source operand indicates whether to use the positive-

MCS-51 Boolean processing instruction subset.

logic value or the logical complement of the addressed bit. (The source operand itself is never affected.)

Bit-test Instructions. The conditional jump instruction "JC rel" (jump on carry) and "JNC rel" (jump on not carry) test the state of the carry flag, branching if it is a one or zero, respectively. (The letters "rel" denote relative code addressing.) The three-byte instruction "JB bit,rel" and "JNB bit,rel" (jump on bit and jump on not bit) test the state of *any* addressable bit in a similar manner. A fifth instruction combines the jump on bit and clear operations. "JBC bit,rel" conditionally branches to the indicated address, then clears the bit in the same two cycle instruction.

All 8051 conditional jump instructions use program counter-relative addressing, and all execute in two cycles. The last instruction byte encodes a signed displacement ranging from -128 to +127. During execution, the CPU adds this value to the incremented program counter to produce the jump destination. Put another way, a conditional jump to the immediately following instruction would encode 00H in the offset byte.

A section of program or subroutine written using only relative jumps to nearby addresses will have the same machine code independent of the code's location. An assembled routine may be repositioned anywhere in memory, even crossing memory page boundaries, without having to modify the program or recompute destination addresses. To facilitate this flexibility, there is an unconditional "short jump" (SJMP) which uses relative addressing as well. Since a programmer would have quite a chore trying to compute relative offset values from one instruction to another, ASM51 automatically computes the displacement needed given only the destination address or label. An error message will alert the programmer if the destination is "out of range."

By combining general purpose bit operations with certain addressable bits, one can "custom build" several hundred useful instructions. All eight bits of the PSW can be tested directly with conditional jump instructions to monitor (among other things) parity and overflow status.

Besides packing enough density to quadruple the memory capacity of the 8048, a new single-chip microcontroller works on bits and bytes.

Microcontroller doubles as Boolean processor

The latest single-chip microcontroller family from Intel brings forth a controller whose dual personality sets it well apart. Based on HMOS, the 8051 (and family) packs enough density to offer four times the memory of the 8048; in addition, the instruction set is much more comprehensive. Not only that, the 8051 can also handle Boolean variables—a major breakthrough for microcomputers that makes the 8051 a byte and bit processor.

Boasting 60,000 transistors, compared to the 8048's 17,000, the 8051 offers 4 kbytes of ROM and 128 bytes of RAM. Other members of the new HMOS family include the 8751 EPROM version, meant for prototyping, and the 8031, which relies on external program memory. All three chips are pin-compatible; they can address 64 kbytes each of program and data memory.

But while the 8051 is a capable processor for 8-bit binary as well as BCD arithmetic, along with 8-bit logic operations, the real standout feature is its ability to handle Boolean variables. Individual bits in special-function registers (SFRs) and 128 software flags can be the operands for logical conditional-branch and transfer operations.

Although integrated with the 8051's architecture (Fig. 1), the Boolean processor may be regarded as an independent bit processor. It has its own instruction set, its own accumulator (the carry flag), its own bit-addressable RAM and its own I/O. The bit-manipulation instructions allow the direct addressing of 128 bits within the internal data RAM and 128 bits within the SFRs.

On any addressable bit, the Boolean processor can perform the following bit operations: Set, Clear, Complement, Jump If Set, Jump If Not Set, Jump If Set Then Clear, and Move To/From Carry. Be-

Bob Koehler, Marketing Product Manager
Intel Microcontroller Operation

Reprinted with permission from *Electronic Design*, Vol. 28, No. 11; copyright Hayden Publishing Co., Inc., 1980.

tween any addressable bit (or its complement) and the carry flag, the Boolean processor can perform the bit operations of AND and OR, the result going into the carry flag.

The bit-manipulation instructions provide excellent code and speed efficiency in bit-intensive applications, such as controlling the 8051's on-chip peripherals. The Boolean processor also provides a direct way to convert into software the logic equations used in random-logic design. Complex combinatorial-logic functions can be resolved without extensive data movement, byte masking and test-and-branch trees (Fig. 2).

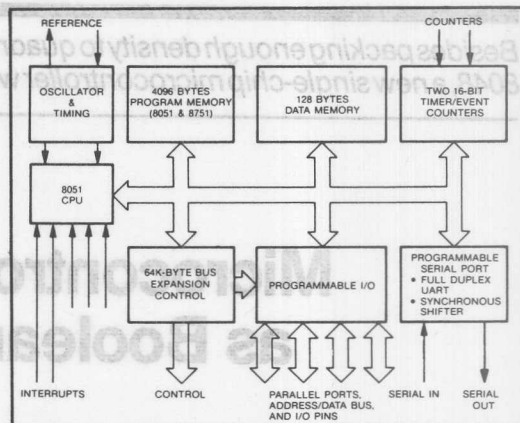
A better architecture

The 8051 CPU manipulates operands in four memory spaces: the 64-kbyte program memory, 64-kbyte external data memory, 16-bit program counter and 384-byte internal data memory, which is further divided into the 256-byte internal data RAM and 128-byte special-function register (Fig. 3). Internal data RAM contains four register banks of eight registers each as well as the stack and 128 addressable bits.

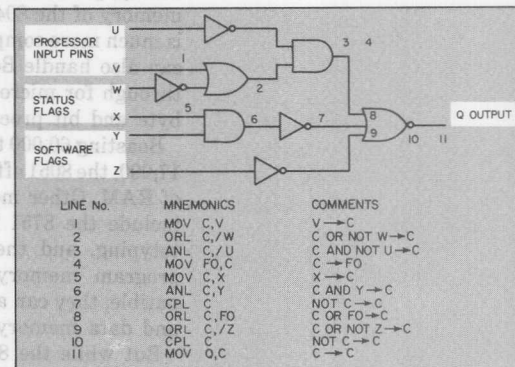
Stack depth is limited only by the available internal data RAM; the starting address is determined by an 8-bit stack pointer. Currently, the lower 128 bytes of internal RAM address space are filled with on-chip RAM—the upper 128 bytes can be added in later products without affecting existing software.

All registers (except for the program counter) reside in the SFR address space. Memory-mapped, they include arithmetic registers, pointers, I/O ports, interrupt system registers, timers and the serial port (Fig. 4).

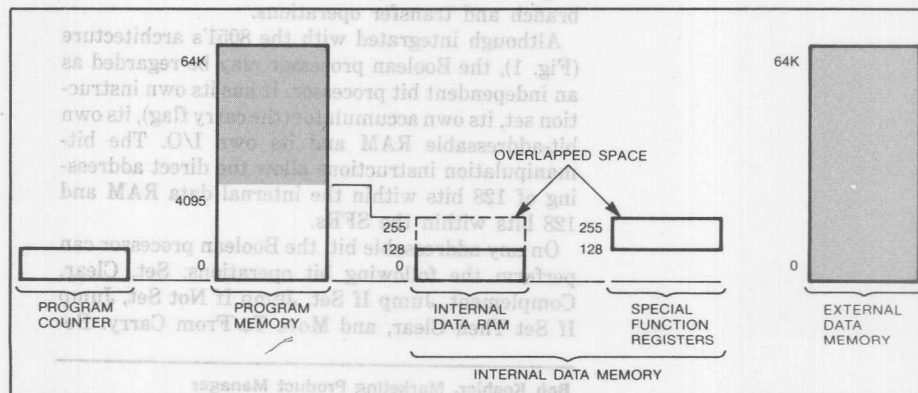
Users can directly address 128 bits in the SFR address space. The 8051 uses only 20 of the 128 bytes in the space for SFRs, so future members of the microcontroller family have room for up to 108 more



1. The architecture of the 8051 family is strongly bus-oriented. On-board counters and a programmable serial port enhance the chip's controller capabilities.



2. The Boolean operations built into the 8051 directly solve problems like this example, which might represent timing control for a keyboard. Variables in the program and Boolean operations are cued to the schematic (white).



3. Both the on-board data and program memories (white) can be supplemented by external RAM to provide about 64 kbytes for both types.

counters, ports, a/d converters and the like, without affecting existing software.

The 8051's program-memory space is not paged and accommodates relocatable code. Conditional branches are offset to the program counter. The register-indirect jump permits branching relative to a 16-bit base register; an 8-bit index register provides offset. Sixteen-bit jumps and calls allow branching to any location within the contiguous 64-kbyte program-memory address space.

Addressing modes for all needs

Source operands can be addressed five ways: *register*, *direct*, *register-indirect*, *immediate* and *base-register plus index-register indirect*. The first three can be used for addressing destination operands. Most instructions have a Destination, Source field that specifies data type, addressing methods and operands. Except for move instructions, the destination operand is also a source operand.

The *register*, *direct*, and *register indirect* addressing modes access registers in the register bank, located in internal data RAM. *Direct* and *register-indirect* modes access the 128 bytes of internal data RAM while *direct* addressing provides access to the SFRs. To access external data memory, *register-indirect* addressing is used, while the fifth mode, *base-register plus index-register indirect*, provides access to lookup tables residing in program memory.

The 8051's internal ROM, RAM, SFRs, ALU and external data bus are all 8 bits wide, but the processor can handle single-bit, 4-bit and 16-bit data types, too. Facilities for 8-bit data transfer, logic and integer-arithmetic operations are supplemented by data transfer, logic and conditional branch operations performed directly on Boolean variables.

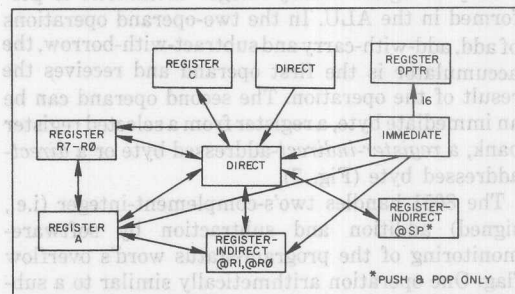
The instruction set, while similar to that of the 8048 family, is enhanced to allow expansion of on-chip CPU peripherals, to sharpen byte efficiency and execution speed, and to contain new high-power operations and new addressing modes that provide a more symmetrical instruction set. It includes 49 single-byte, 45 double-byte and 17 triple-byte instructions. At a 12-MHz clock rate, 64 of the instructions execute in 1 μ s and 45 in 2 μ s while multiplications and divisions take 4 μ s.

Comprehensive Move instructions

The 8051 boasts other improvements as well. For one thing, data transfer, logic manipulation, arithmetic processing and real-time control capabilities have all been enhanced. Lookup tables, residing in program memory, can be accessed by indirect moves, which enables a byte to be transferred from the location whose address is the sum of a 16-bit base register and the 8-bit index register. This feature is

Arithmetic registers	: Accumulator, B register, program-status word
Pointers	: Stack pointer, data pointer (high and low)
Parallel-I/O ports	: Port 0, port 1, port 2, port 3
Interrupt system	: Interrupt-priority control, interrupt-enable control
Timers	: Timer mode, timer control, timers 0 and 1 (high and low)
Serial-I/O port	: Serial control, serial-data buffer

4. Special-function registers, shown in Fig. 3, are used for arithmetic, I/O, timers and the interrupt system.



5. Move operations are possible from nearly any storage location to nearly any other. Except where noted, all moves operate on bytes.

convenient for programming translation algorithms such as ASCII-to-seven-segment conversions.

A byte location within a 256-byte block of external data memory can be accessed through *register-indirect* addressing via an 8-bit base register. Furthermore, any location within the full 64-kbyte external data memory address space can be accessed through *register-indirect* addressing via a 16-bit base register.

Byte-constant moves (or immediate moves) and byte-variable moves are highly orthogonal (Fig. 5). In other words, a byte operand located anywhere in the internal data memory can be transferred to nearly any other location in the memory by a single instruction, as can immediate operands. The direct-address-to-direct-address move permits the value in a port to be moved to the internal data RAM without using any registers or the accumulator.

In addition, the accumulator can be exchanged with a register in a selected register bank, with a *register-indirect*-addressed byte in the internal data RAM, or with a *direct*-addressed byte in the internal data RAM or SFR. The least significant half-byte of the accumulator and of a *register-indirect*-addressed byte in internal data RAM can also be exchanged.

The 8051 permits a second operand to perform the logic operations of AND, OR and XOR on the accumulator. This second operand can be an immediate value, a register in a selected register bank, a *register-indirect*-addressed byte of internal data RAM, a *direct*-addressed byte of internal data RAM

Boolean microcontroller

or an SFR.

The three logic operations can also be performed on a *direct*-addressed byte of internal data RAM or an SFR, using the accumulator as the second operand. Any bit anywhere in the internal data RAM or SFRs can be set, cleared or complemented using the logic operations with *immediate*-addressing to *direct* addressing (Fig. 6). The remaining logic operations work only on the accumulator.

Only unsigned binary-integer arithmetic is performed in the ALU. In the two-operand operations of add, add-with-carry and subtract-with-borrow, the accumulator is the first operand and receives the result of the operation. The second operand can be an immediate byte, a register from a selected register bank, a *register-indirect*-addressed byte or a *direct*-addressed byte (Fig. 7).

The 8051 handles two's-complement-integer (i.e., signed) addition and subtraction by software-monitoring of the program-status word's overflow flag. One operation arithmetically similar to a subtraction is the instruction Compare And Jump If Not Equal, which performs a conditional branch if a register in a selected bank (or an *indirect*-addressed byte of internal data RAM) does not equal an immediate value, or if the accumulator does not equal a byte of *direct*-addressable internal data RAM or an SFR.

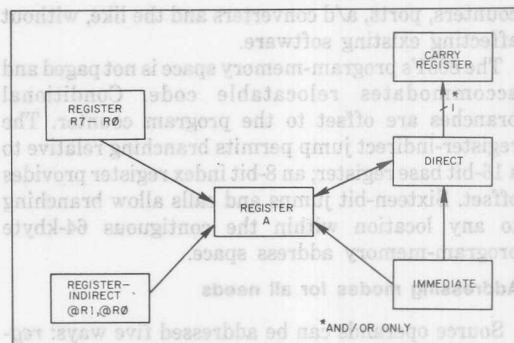
Three arithmetic functions operate exclusively on the accumulator: Decimal Adjust For Addition, Jump If Accumulator Is Zero and Jump If Accumulator Is Not Zero. Conditional branches may be taken, based on the value in the accumulator, whether zero or not.

Software counters are easy to implement with the 8051 because increment and decrement operations can be performed on the accumulator, a register in a selected register bank, an *indirect*-addressed byte in the internal data RAM, or a byte in the *direct*-addressed internal RAM or SFR. The operation Decrement And Jump If Not Zero facilitates efficient loop control. This operation can test a register in a selected register bank, any SFR, or any byte or internal data RAM accessible through *direct* addressing, and then force a branch if it is not zero. In addition, the 16-bit data pointer can be incremented.

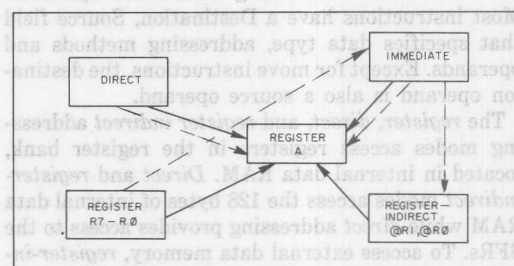
Peripherals galore

Counting, timing and communications in a real-time, priority-oriented environment are the keystones of control applications. External events and peripherals require asynchronous servicing by the CPU. A sophisticated nested interrupt system ties these asynchronous activities to regular program execution with response latencies of 3 to 7 μ s.

The 8051 services interrupt requests coming from



6. Logic operations include many source and destination locations and—except where noted—AND, OR and XOR.



7. Various operations involve different registers: Add, Add With Carry and Subtract With Borrow are indicated by black lines, Compare And Jump If Not Equal by broken lines.

five sources—two via the INT 0 and INT 1 pins, one from each of the two internal counters and one from the serial-I/O port. Each vectored interrupt accesses a location in program memory for its service routine. Each of the five interrupt sources can not only be assigned to one of two priority levels but can also be enabled and disabled independently. Finally, all enabled sources can be disabled or enabled globally (Fig. 8).

The 8051 has instructions that treat its 32 I/O lines as 32 individually addressable bits or as four parallel 8-bit ports (Fig. 9). Each pin of port 0 can be configured as an open drain output or as a high-impedance input. Ports 1, 2, and 3 are quasi-bidirectional buffers that can sink or source one TTL load. Port 0 also provides a microprocessor bus with multiplexed low-order address and data; it serves as an interface with standard MCS-85 memories and MCS-80 peripherals. Port 2 provides the high-order address bus needed to expand the 8051 with external program or data memory. The bus driver can sink or source two TTL loads.

At 12 MHz, the program-memory cycle is 500 ns and the access times from stable address and program-store enable are approximately 320 ns and 150 ns, respectively. For external memory, the cycle

is 1 μ s and the corresponding access times are approximately 600 ns and 250 ns (pins \overline{RD} and \overline{WR} of port 3).

The remaining pins of port 3 can be used for other functions: Interrupt-request inputs (INT0 and INT1), counter inputs (T0 and T1) and the serial port's receiver input (RXD) and transmitter output (TXD).

Timing is essential

The 8051 contains two 16-bit counters for measuring time intervals, counting events, measuring pulse widths and generating periodic interrupt requests—a big help in control applications. Each can be programmed to operate in one of three modes: Mode 0 is similar to an 8038's 8-bit timer with prescaler and 8-bit event counter; mode 1 is a 16-bit time-interval or event counter; and mode 2 is an 8-bit time-interval or event counter with automatic reload upon overflow. Counter 0 can also be programmed in a fourth mode that divides it into one 8-bit time interval or event counter and one 8-bit time-interval counter.

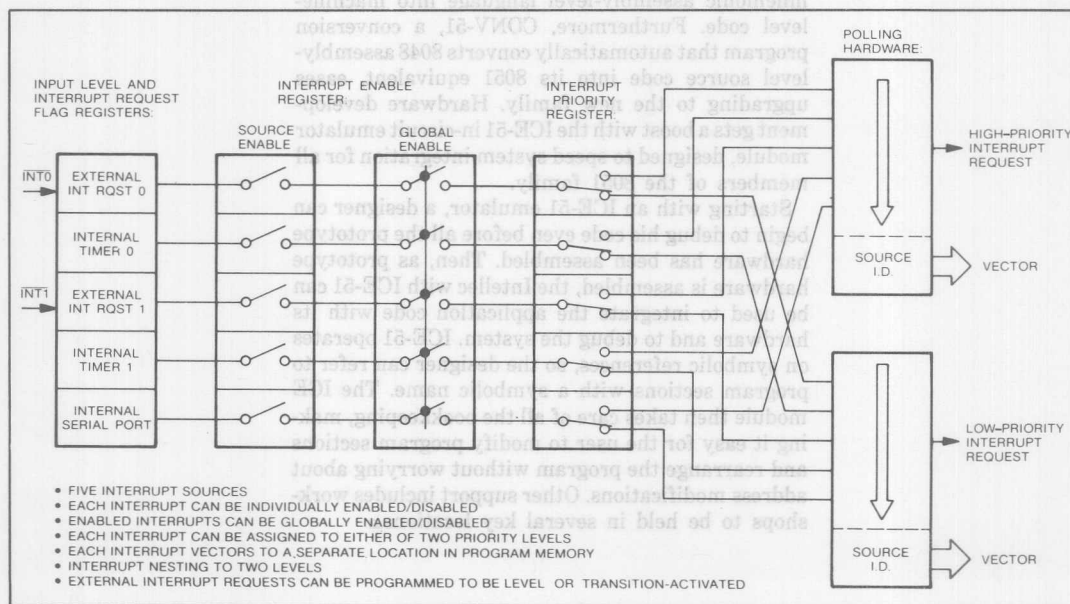
The 8051's counters not only offer flexibility and 16-bit precision, they also can handle very high input frequencies. For external inputs, the frequency range is from 0 Hz to 0.5 MHz; when programmed for an input of the internal oscillator, they cover 0.1 to 1 MHz. To measure pulse width directly, an external input can be gated to the counter by a second external source. The counters are started and

stopped under software control. Each counter sets its interrupt request flag when it overflows from all ONES to all ZEROS, or to the auto-reload value.

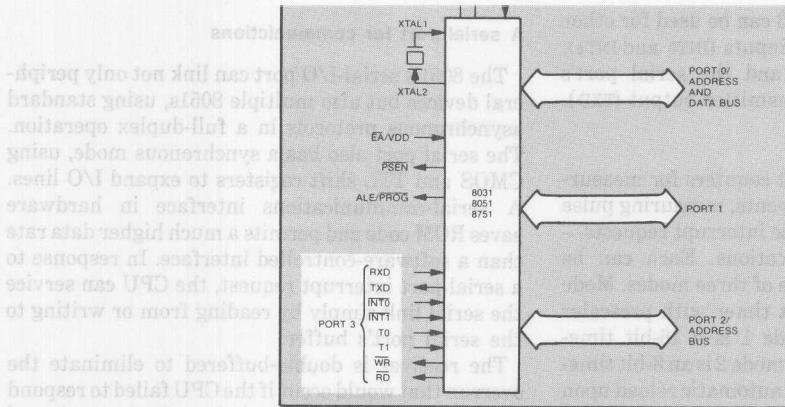
A serial port for communications

The 8051's serial I/O port can link not only peripheral devices but also multiple 8051s, using standard asynchronous protocols in a full-duplex operation. The serial port also has a synchronous mode, using CMOS and TTL shift registers to expand I/O lines. A serial-communications interface in hardware saves ROM code and permits a much higher data rate than a software-controlled interface. In response to a serial-port interrupt request, the CPU can service the serial link simply by reading from or writing to the serial port's buffer.

The receiver is double-buffered to eliminate the overrun that would occur if the CPU failed to respond to the receiver's interrupt before the beginning of the next frame. Furthermore, the 8051 offers false-start-bit rejection on received frames and, for noise rejection, takes a best-two-out-of-three poll on three samples near the center of each received bit. When interfacing with standard UART devices, the serial channel can be programmed to transmit or receive a 10-bit frame at communication rates of 122 to 31,250 baud; 11-bit frames can, in addition, be transmitted at 187,500 baud. The interprocessor-communications mode is similar to the two standard UART modes, but provides automatic wakeup of



8. The 8051 family's interrupt system is unusually versatile. Global and individual enables can be combined with both software and hardware-priority assignments.



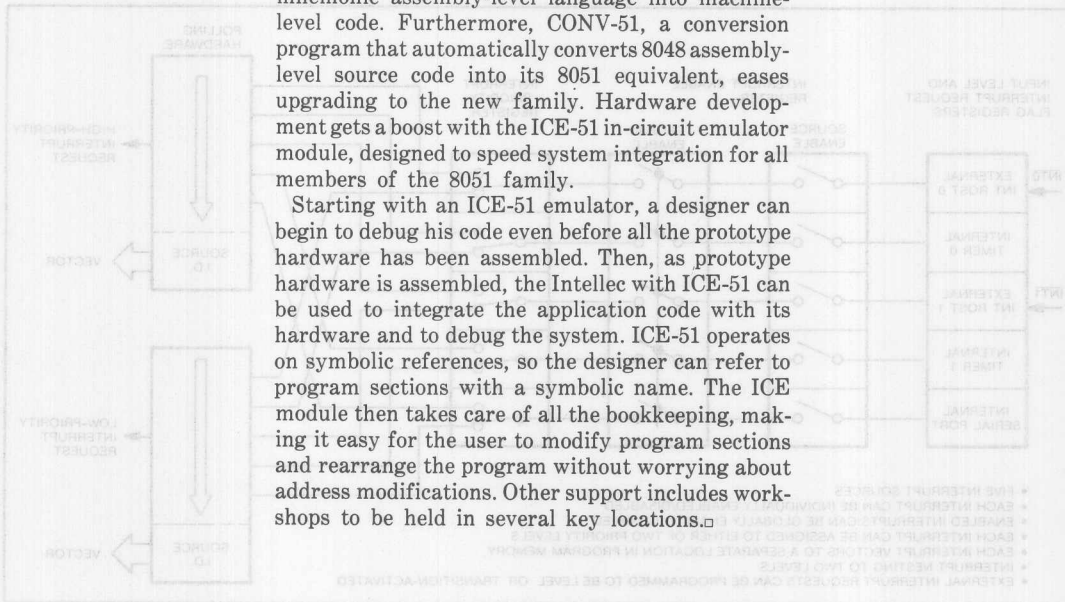
9. Four 8-bit ports (white) take up most of the 8051's 40 pins. The data bus is split between port 0 and port 2.

slave processors.

Even the best μC is in trouble without development support, but the 8051 family, new as it is, can count on Intel's line of Intellect development systems. Support software, development systems and a user library stocked with 8051 utility and applications programs also support software development.

Users can write 8051 code with ASM-51, an absolute macroassembler, that converts the mnemonic assembly-level language into machine-level code. Furthermore, CONV-51, a conversion program that automatically converts 8048 assembly-level source code into its 8051 equivalent, eases upgrading to the new family. Hardware development gets a boost with the ICE-51 in-circuit emulator module, designed to speed system integration for all members of the 8051 family.

Starting with an ICE-51 emulator, a designer can begin to debug his code even before all the prototype hardware has been assembled. Then, as prototype hardware is assembled, the Intellect with ICE-51 can be used to integrate the application code with its hardware and to debug the system. ICE-51 operates on symbolic references, so the designer can refer to program sections with a symbolic name. The ICE module then takes care of all the bookkeeping, making it easy for the user to modify program sections and rearrange the program without worrying about address modifications. Other support includes workshops to be held in several key locations.



An Introduction to the Intel MCS-51[®] Single-Chip Microcomputer Family

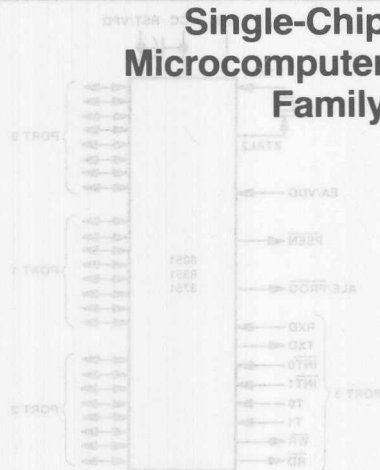


Figure 1. Intel MCS-51 Single-Chip Microcomputer Logic Symbol

microprocessor (previously Intel), of course) or have a background in computer programming and digital logic.

Family Overview

Pin diagrams for the 8051, 8751, and 8031 are shown in Figure 1. The devices include the following features:

- Single-supply 5 volt operation using HMOS technology.
- 4096 bytes program memory on-chip (not on 8031).
- 128 bytes data memory on-chip.
- Four register banks.
- 128 user-defined software flags.
- 64 Kilobytes each program and external RAM addressability.
- One microsecond instruction cycle with 12 MHz crystal.
- 32 bidirectional I/O lines organized as four 8-bit ports (16 lines on 8031).
- Multiple modes, high-speed programmable Serial Port.
- Two multiple mode, 16-bit Timer/Counter.
- Two-level prioritized interrupt structure.
- Full depth stack for subroutine return linkage and data storage.
- Augmented MCS-48[™] instruction set.
- Direct Byte and bit addressability.
- Binary or Decimal arithmetic.
- Signed-overflow detection and parity computation.
- Hardware Multiply and Divide in 4 μ sec.
- Integrated Boolean Processor for control applications.
- Upwardly compatible with existing 8048 software.

Contents

1. INTRODUCTION	2-14
Family Overview	2-14
Microcomputer Background Concepts	2-15
2. ARCHITECTURE AND ORGANIZATION	2-17
Central Processing Unit	2-18
Memory Spaces	2-21
Input/Output Ports	2-22
Special Peripheral Functions	2-23
3. INSTRUCTION SET AND ADDRESSING MODES	2-27
Data Addressing Modes	2-27
Addressing Mode Combinations	2-30
Advantages of Symbolic Addressing	2-30
Arithmetic Instruction Usage	2-31
Multiplication and Division	2-32
Logical Byte Operations	2-32
Program Control	2-33
Operate-and-Branch Instructions	2-34
Stack Operations	2-34
Table Look-Up Instructions	2-35
4. BOOLEAN PROCESSING INSTRUCTIONS	2-37
Direct Bit Addressing	2-37
Bit Manipulation Instructions	2-37
Solving Combinatorial Logic Equations	2-38
5. ON-CHIP PERIPHERAL FUNCTIONS	2-40
I/O Ports	2-40
Serial Port and Timer	2-41
6. SUMMARY	2-42

automobile engine to test or data processing equipment. Follow-on products stretched the MCS-48[™] architecture in several directions: the 8049 and 8039 doubled the amount of on-chip memory and ran 8MHz faster; the 8051 reduced costs by executing a subset of the 8048 instructions with a somewhat slower clock; and the 8032 put a unique two-channel 8-bit analog-to-digital converter on the same HMOS chip as the computer, leaving the chip interface directly with analog transducers.

Now three new high-performance single-chip microcomputers—the Intel[®] 8051, 8751, and 8031—extend the advantages of integrated Electronics to whole new product areas. Thanks to Intel's new HMOS technology, the MCS-51[™] family provides four times the program memory and twice the data memory as the 8048 on a single chip. New I/O and peripheral capabilities both increase the range of applicability and reduce total system cost. Depending on the use, processing throughput increases by two and one-half to ten times.

This Application Note is intended to introduce the reader to the MCS-51[™] architecture and features. While it does not assume intimacy with the MCS-48[™] product line on the part of the reader, he/she should be familiar with

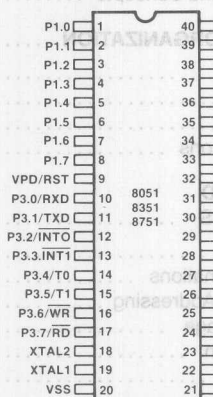


Figure 1a. 8051 Microcomputer Pinout Diagram

1. INTRODUCTION

In 1976 Intel introduced the MCS-48™ family, consisting of the 8048, 8748, and 8035 microcomputers. These parts marked the first time a complete microcomputer system, including an eight-bit CPU, 1024 8-bit words of ROM or EPROM program memory, 64 words of data memory, I/O ports and an eight-bit timer/counter could be integrated onto a single silicon chip. Depending only on the program memory contents, one chip could control a limitless variety of products, ranging from appliances or automobile engines to text or data processing equipment. Follow-on products stretched the MCS-48™ architecture in several directions: the 8049 and 8039 doubled the amount of on-chip memory and ran 83% faster; the 8021 reduced costs by executing a subset of the 8048 instructions with a somewhat slower clock; and the 8022 put a unique two-channel 8-bit analog-to-digital converter on the same NMOS chip as the computer, letting the chip interface directly with analog transducers.

Now three new high-performance single-chip microcomputers—the Intel® 8051, 8751, and 8031—extend the advantages of Integrated Electronics to whole new product areas. Thanks to Intel's new HMOS technology, the MCS-51™ family provides four times the program memory and twice the data memory as the 8048 on a single chip. New I/O and peripheral capabilities both increase the range of applicability and reduce total system cost. Depending on the use, processing throughput increases by two and one-half to ten times.

This Application Note is intended to introduce the reader to the MCS-51™ architecture and features. While it does not assume intimacy with the MCS-48™ product line on the part of the reader, he/she should be familiar with

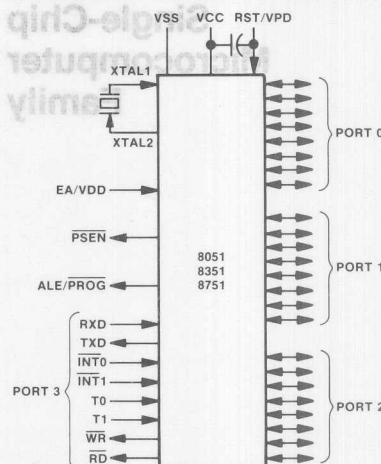


Figure 1b. 8051 Microcomputer Logic Symbol

some microprocessor (preferably Intel's, of course) or have a background in computer programming and digital logic.

Family Overview

Pinout diagrams for the 8051, 8751, and 8031 are shown in Figure 1. The devices include the following features:

- Single-supply 5 volt operation using HMOS technology.
- 4096 bytes program memory on-chip (not on 8031).
- 128 bytes data memory on-chip.
- Four register banks.
- 128 User-defined software flags.
- 64 Kilobytes each program and external RAM addressability.
- One microsecond instruction cycle with 12 MHz crystal.
- 32 bidirectional I/O lines organized as four 8-bit ports (16 lines on 8031).
- Multiple mode, high-speed programmable Serial Port.
- Two multiple mode, 16-bit Timer/Counters.
- Two-level prioritized interrupt structure.
- Full depth stack for subroutine return linkage and data storage.
- Augmented MCS-48™ instruction set.
- Direct Byte and Bit addressability.
- Binary or Decimal arithmetic.
- Signed-overflow detection and parity computation.
- Hardware Multiple and Divide in 4 μ sec.
- Integrated Boolean Processor for control applications.
- Upwardly compatible with existing 8048 software.

All three devices come in a standard 40-pin Dual In-Line Package, with the same pin-out, the same timing, and the same electrical characteristics. The primary difference between the three is the on-chip program memory—different types are offered to satisfy differing user requirements.

The 8751 provides 4K bytes of ultraviolet-Erasable, Programmable Read Only Memory (EPROM) for program development, prototyping, and limited production runs. (By convention, 1K means $2^{10} = 1024$. 1k—with a lower case “k”—equals $10^3 = 1000$.) This part may be individually programmed for a specific application using Intel's Universal PROM Programmer (UPP). If software bugs are detected or design specifications change the same part may be “erased” in a matter of minutes by exposure to ultraviolet light and reprogrammed with the modified code. This cycle may be repeated indefinitely during the design and development phase.

The final version of the software must be programmed into a large number of production parts. The 8051 has 4K bytes of ROM which are mask-programmed with the customer's order when the chip is built. This part is considerably less expensive, but cannot be erased or altered after fabrication.

The 8031 does not have any program memory on-chip, but may be used with up to 64K bytes of external standard or multiplexed ROMs, PROMs, or EPROMs. The 8031 fits well in applications requiring significantly larger or smaller amounts of memory than the 4K bytes provided by its two siblings.

(The 8051 and 8751 automatically access external program memory for all addresses greater than the 4096 bytes on-chip. The External Access input is an override for all internal program memory—the 8051 and 8751 will each emulate an 8031 when pin 31 is low.)

Throughout this Note, “8051” is used as a generic term. Unless specifically stated otherwise, the point applies equally to all three components. Table 1 summarizes the quantitative differences between the members of the MCS-51™ and MCS-51™ families.

The remainder of this Note discusses the various MCS-51™ features and how they can be used. Software and/or hard-

ware application examples illustrate many of the concepts. Several isolated tasks (rather than one complete system design example) are presented in the hope that some of them will apply to the reader's experiences or needs.

A document this short cannot detail all of a computer system's capabilities. By no means will all the 8051 instructions be demonstrated; the intent is to stress new or unique MCS-51™ operations and instructions generally used in conjunction with each other. For additional hardware information refer to the Intel MCS-51™ Family User's Manual, publication number 121517. The assembly language and use of ASM51, the MCS-51™ assembler, are further described in the MCS-51™ Macro Assembler User's Guide, publication number 9800937.

The next section reviews some of the basic concepts of microcomputer design and use. Readers familiar with the 8048 may wish to skim through this section or skip directly to the next, “ARCHITECTURE AND ORGANIZATION.”

Microcomputer Background Concepts

Most digital computers use the binary (base 2) number system internally. All variables, constants, alphanumeric characters, program statements, etc., are represented by groups of binary digits (“bits”), each of which has the value 0 or 1. Computers are classified by how many bits they can move or process at a time.

The MCS-51™ microcomputers contain an eight-bit central processing unit (CPU). Most operations process variables eight bits wide. All internal RAM and ROM, and virtually all other registers are also eight bits wide. An eight-bit (“byte”) variable (shown in Figure 2) may assume one of $2^8 = 256$ distinct values, which usually represent integers between 0 and 255. Other types of numbers, instructions, and so forth are represented by one or more bytes using certain conventions.

For example, to represent positive and negative values, the most significant bit (D7) indicates the sign of the other seven bits—0 if positive, 1 if negative—allowing integer variables between -128 and +127. For integers with extremely large magnitudes, several bytes are manipulated together as “multiple precision” signed or unsigned integers—16, 24, or more bits wide.

Table 1. Features of Intel's Single-Chip Microcomputers

EPROM Program Memory	ROM Program Memory	External Program Memory	Program Memory (Int/Max)	Data Memory (Bytes)	Instr. Cycle Time	Input/Output Pins	Interrupt Sources	Reg. Banks
—	8021		1K/1K	64	8.4 μ Sec	21	0	1
	8022		2K/2K	64	8.4 μ Sec	28	2	1
8748	8048	8035	1K/4K	64	2.5 μ Sec	27	2	2
	8049	8039	2K/4K	128	1.36 μ Sec	27	2	2
8751	8051	8031	4K/64K	128	1.0 μ Sec	32	5	4

The letters "MCS" have traditionally indicated a system or family of compatible Intel® microcomputer components, including CPUs, memories, clock generators, I/O expanders, and so forth. The numerical suffix indicates the microprocessor or microcomputer which serves as the cornerstone of the family. Microcomputers in the MCS-48™ family currently include the 8048-series (8035, 8048, & 8748), the 8049-series (8039 & 8049), and the 8021 and 8022; the family also includes the 8243, an I/O expander compatible with each of the microcomputers. Each computer's CPU is derived from the 8048, with essentially the same architecture, addressing modes, and instruction set, and a single assembler (ASM48) serves each.

The first members of the MCS-51™ family are the 8051, 8751, and 8031. The architecture of the 8051-series, while derived from the 8048, is not strictly compatible; there are more addressing modes, more instructions, larger address spaces, and a few other hardware differences. In this Application Note the letters "MCS-51" are used when referring to architectural features of the 8051-series—features which would be included on possible future microcomputers based on the 8051 CPU. Such products could have different amounts of memory (as in the 8048/8049) or different peripheral functions (as in the 8021 and 8022) while leaving the CPU and instruction set intact. ASM51 is the assembler used by all microcomputers in the 8051 family.

Two digit decimal numbers may be "packed" in an eight-bit value, using four bits for the binary code of each digit. This is called Binary-Coded Decimal (BCD) representation, and is often used internally in programs which interact heavily with human beings.

Alphanumeric characters (letters, numbers, punctuation marks, etc.) are often represented using the American Standard Code for Information Interchange (ASCII) convention. Each character is associated with a unique seven-bit binary number. Thus one byte may represent

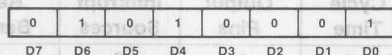


Figure 2. Representation of Bits Within an Eight-Bit "Byte" (Value shown = 01010001 Binary = 81 decimal).

a single character, and a word or sequence of letters may be represented by a series (or "string") of bytes. Since the ASCII code only uses 128 characters, the most significant bit of the byte is not needed to distinguish between characters. Often D7 is set to 0 for all characters. In some coding schemes, D7 is used to indicate the "parity" of the other seven bits—set or cleared as necessary to ensure that the total number of "1" bits in the eight-bit code is even ("even parity") or odd ("odd parity"). The 8051 includes hardware to compute parity when it is needed.

A computer program consists of an ordered sequence of specific, simple steps to be executed by the CPU one-at-a-time. The method or sequence of steps used collectively to solve the user's application is called an "algorithm."

The program is stored inside the computer as a sequence of binary numbers, where each number corresponds to one of the basic operations ("opcodes") which the CPU is capable of executing. In the 8051, each program memory location is one byte. A complete instruction consists of a sequence of one or more bytes, where the first defines the operation to be executed and additional bytes (if needed) hold additional information, such as data values or variable addresses. No instruction is longer than three bytes.

The way in which binary opcodes and modifier bytes are assigned to the CPU's operations is called the computer's "machine language." Writing a program directly in machine language is time-consuming and tedious. Human beings think in words and concepts rather than encoded numbers, so each CPU operation and resource is given a name and standard abbreviation ("mnemonic"). Programs are more easily discussed using these standard mnemonics, or "assembly language," and may be typed into an Intel® Intellect® 800 or Series II® microcomputer development system in this form. The development system can mechanically translate the program from assembly language "source" form to machine language "object" code using a program called an "assembler." The MCS-51™ assembler is called ASM51.

There are several important differences between a computer's machine language and the assembly language used as a tool to represent it. The machine language or instruction set is the set of operations which the CPU can perform while a program is executing ("at run-time"), and is strictly determined by the microcomputer hardware design.

The assembly language is a standard (though more-or-less arbitrary) set of symbols including the instruction set mnemonics, but with additional features which further simplify the program design process. For example, ASM51 has controls for creating and formatting a program listing, and a number of directives for allocating variable storage and inserting arbitrary bytes of data into the object code for creating tables of constants.

In addition, ASM51 can perform sophisticated mathematical operations, computing addresses or evaluating arithmetic expressions to relieve the programmer from this drudgery. However, these calculations can only use information known at "assembly time."

For example, the 8051 performs arithmetic calculations at run-time, eight bits at a time. ASM51 can do similar operations 16 bits at a time. The 8051 can only do one simple step per instruction, while ASM51 can perform complex calculations in each line of source code. However, the operations performed by the assembler may only use parameter values fixed at assembly-time, not variables whose values are unknown until program execution begins.

For example, when the assembly language source line,

```
ADD A,#(LOOP_COUNT + 1) * 3
```

is assembled, ASM51 will find the value of the previously-defined constant "LOOP_COUNT" in an internal symbol table, increment the value, multiply the sum by three, and (assuming it is between -256 and 255 inclusive) truncate the product to eight bits. When this instruction is executed, the 8051 ALU will just add that resulting constant to the accumulator.

Some similar differences exist to distinguish number system ("radix") specifications. The 8051 does all computations in binary (though there are provisions for then converting the result to decimal form). In the course of writing a program, though, it may be more convenient to specify constants using some other radix, such as base 10. On other occasions, it is desirable to specify the ASCII code for some character or string of characters without referring to tables. ASM51 allows several representations for constants, which are converted to binary as each instruction is assembled.

For example, binary numbers are represented in the

assembly language by a series of ones and zeros (naturally), followed by the letter "B" (for Binary); octal numbers as a series of octal digits (0-7) followed by the letter "O" (for Octal) or "Q" (which doesn't stand for anything, but looks sort of like an "O" and is less likely to be confused with a zero).

Hexadecimal numbers are represented by a series of hexadecimal digits (0-9, A-F), followed by (you guessed it) the letter "H." A "hex" number must begin with a decimal digit; otherwise it would look like a user-defined symbol (to be discussed later). A "dummy" leading zero may be inserted before the first digit to meet this constraint. The character string "BACH" could be a legal label for a Baroque music synthesis routine; the string "0BACH" is the hexadecimal constant BACH₁₆. This is a case where adding 0 makes a big difference.

Decimal numbers are represented by a sequence of decimal digits, optionally followed by a "D." If a number has no suffix, it is assumed to be decimal—so it had better not contain any non-decimal digits. "0BAC" is not a legal representation for anything.

When an ASCII code is needed in a program, enclose the desired character between two apostrophes (as in '#') and the assembler will convert it to the appropriate code (in this case '23H'). A string of characters between apostrophes is translated into a series of constants; 'BACH' becomes 42H, 41H, 43H, 48H.

These same conventions are used throughout the associated Intel documentation. Table 2 illustrates some of the different number formats.

2. ARCHITECTURE AND ORGANIZATION

Figure 3 blocks out the MCS-51™ internal organization. Each microcomputer combines a Central Processing Unit, two kinds of memory (data RAM plus program ROM or EPROM), Input/Output ports, and the mode,

Table 2. Notations Used to Represent Numbers

Bit Pattern	Binary	Octal	Hexa-Decimal	Decimal	Signed Decimal
0 0 0 0 0 0 0 0	0B	0Q	00H	0	0
0 0 0 0 0 0 0 1	1B	1Q	01H	1	+1
...
0 0 0 0 0 1 1 1	111B	7Q	07H	7	+7
0 0 0 0 1 0 0 0	1000B	10Q	08H	8	+8
0 0 0 0 1 0 0 1	1001B	11Q	09H	9	+9
0 0 0 0 1 0 1 0	1010B	12Q	0AH	10	+10
...
0 0 0 0 1 1 1 1	1111B	17Q	0FH	15	+15
0 0 0 1 0 0 0 0	10000B	20Q	10H	16	+16
...
0 1 1 1 1 1 1 1	1111111B	177Q	7FH	127	+127
1 0 0 0 0 0 0 0	10000000B	200Q	80H	128	-128
1 0 0 0 0 0 0 1	10000001B	201Q	81H	129	-127
...
1 1 1 1 1 1 1 0	11111110B	376Q	0FEH	254	-2
1 1 1 1 1 1 1 1	11111111B	377Q	0FFH	255	-1

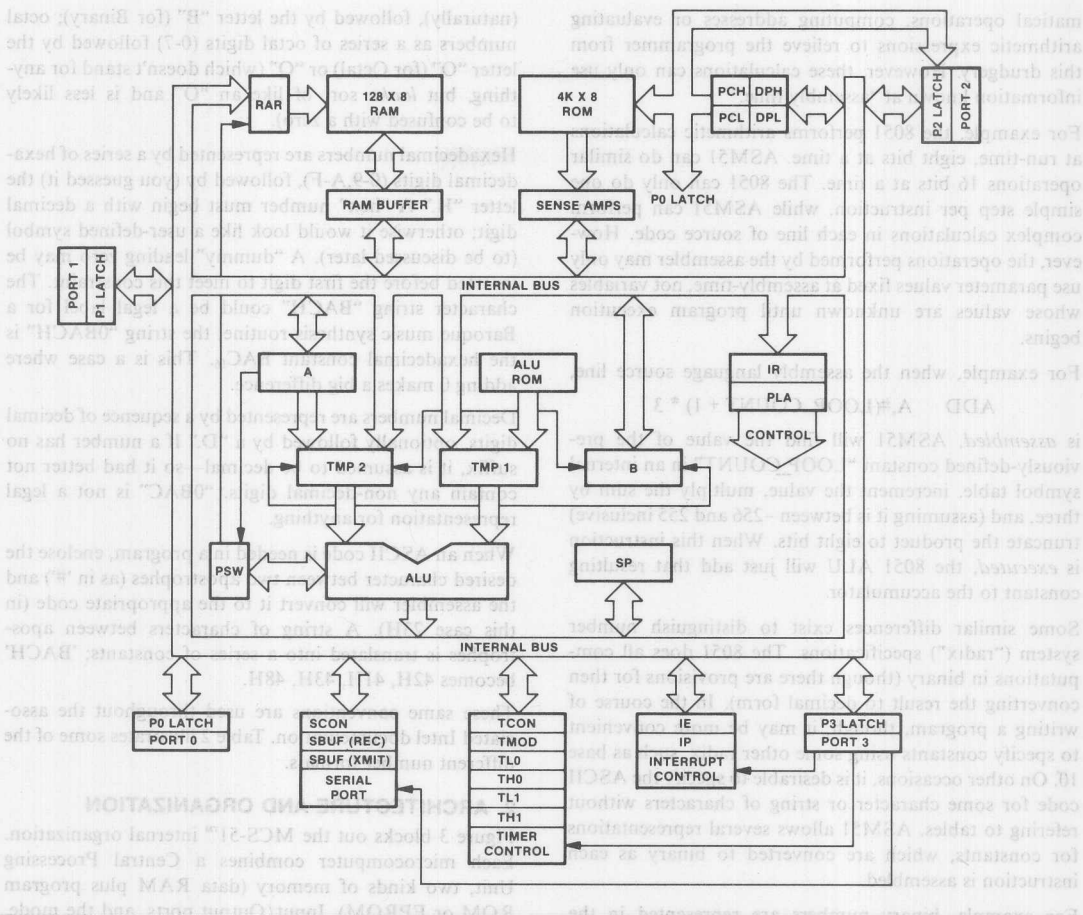


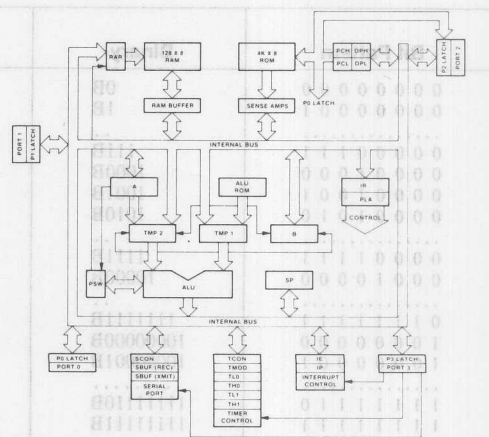
Figure 3. Block Diagram of 8051 Internal Structure

status, and data registers and random logic needed for a variety of peripheral functions. These elements communicate through an eight-bit data bus which runs throughout the chip, somewhat akin to indoor plumbing. This bus is buffered to the outside world through an I/O port when memory or I/O expansion is desired.

Let's summarize what each block does; later chapters dig into the CPU's instruction set and the peripheral registers in much greater detail.

Central Processing Unit

The CPU is the "brains" of the microcomputer, reading the user's program and executing the instructions stored therein. Its primary elements are an eight-bit Arithmetic/Logic Unit with associated registers A, B, PSW, and SP, and the sixteen-bit Program Counter and "Data Pointer" registers.



Arithmetic Logic Unit

The ALU can perform (as the name implies) arithmetic and logic functions on eight-bit variables. The former include basic addition, subtraction, multiplication, and division; the latter include the logical operations AND, OR, and Exclusive-OR, as well as rotate, clear, complement, and so forth. The ALU also makes conditional branching decisions, and provides data paths and temporary registers used for data transfers within the system. Other instructions are built up from these primitive functions: the addition capability can increment registers or automatically compute program destination addresses; subtraction is also used in decrementing or comparing the magnitude of two variables.

These primitive operations are automatically cascaded and combined with dedicated logic to build complex instructions such as incrementing a sixteen-bit register pair. To execute one form of the compare instruction, for example, the 8051 increments the program counter three times, reads three bytes of program memory, computes a register address with logical operations, reads internal data memory twice, makes an arithmetic comparison of two variables, computes a sixteen-bit destination address, and decides whether or not to make a branch—all in two microseconds!

An important and unique feature of the MCS-51 architecture is that the ALU can also manipulate one-bit as well as eight-bit data types. Individual bits may be set, cleared, or complemented, moved, tested, and used in logic computations. While support for a more primitive data type may initially seem a step backwards in an era of increasing word length, it makes the 8051 especially well suited for controller-type applications. Such algorithms *inherently* involve Boolean (true/false) input and output variables, which were heretofore difficult to implement with standard microprocessors. These features are collectively referred to as the MCS-51™ “Boolean Processor,” and are described in the so-named chapter to come.

Thanks to this powerful ALU, the 8051 instruction set fares well at both real-time control and data intensive algorithms. A total of 51 separate operations move and manipulate three data types: Boolean (1-bit), byte (8-bit), and address (16-bit). All told, there are eleven addressing modes—seven for data, four for program sequence control (though only eight are used by more than just a few specialized instructions). Most operations allow several addressing modes, bringing the total number of instructions (operation/addressing mode combinations) to 111, encompassing 255 of the 256 possible eight-bit instruction opcodes.

Instruction Set Overview

Table 4 lists these 111 instructions classified into five groups:

- Arithmetic Operations
- Logical Operations for Byte Variables
- Data Transfer Instructions
- Boolean Variable Manipulation
- Program Branching and Machine Control

MCS-48™ programmers perusing Table 4 will notice the absence of special categories for Input/Output, Timer/Counter, or Control instructions. These functions are all still provided (and indeed many new functions are added), but as special cases of more generalized operations in other categories. To explicitly list all the useful instructions involving I/O and peripheral registers would require a table approximately four times as long.

Observant readers will also notice that all of the 8048's page-oriented instructions (conditional jumps, JMPP, MOVP, MOVP3) have been replaced with corresponding but non-paged instructions. The 8051 instruction set is entirely *non*-page-oriented. The MCS-48™ “MOVP” instruction replacement and all conditional jump instructions operate relative to the program counter, with the actual jump address computed by the CPU during instruction execution. The “MOVP3” and “JMPP” replacements are now made relative to another sixteen-bit register, which allows the effective destination to be anywhere in the program memory space, regardless of where the instruction itself is located. There are even three-byte jump and call instructions allowing the destination to be *anywhere* in the 64K program address space.

The instruction set is designed to make programs efficient both in terms of code size and execution speed. No instruction requires more than three bytes of program memory, with the majority requiring only one or two bytes. Virtually all instructions execute in either one or two instruction cycles—one or two microseconds with a 12-MHz crystal—with the sole exceptions (multiply and divide) completing in four cycles.

Many instructions such as arithmetic and logical functions or program control, provide both a short and a long form for the same operation, allowing the programmer to optimize the code produced for a specific application. The 8051 usually fetches two instruction bytes per instruction cycle, so using a shorter form can lead to faster execution as well.

For example, any byte of RAM may be loaded with a constant with a three-byte, two-cycle instruction, but the commonly used “working registers” in RAM may be initialized in one cycle with a two-byte form. Any bit anywhere on the chip may be set, cleared, or complemented by a single three-byte logical instruction using two cycles. But critical control bits, I/O pins, and software flags may be controlled by two-byte, single cycle instructions. While three-byte jumps and calls can “go anywhere” in program memory, nearby sections of code may be reached by shorter relative or absolute versions.

(MSB)				(LSB)			
CY	AC	F0	RS1	RS0	OV	—	P

Symbol Position Name and Significance

CY	PSW.7	Carry flag.
		Set/cleared by hardware or software during certain arithmetic and logical instructions.
AC	PSW.6	Auxiliary Carry flag.
		Set/cleared by hardware during addition or subtraction instructions to indicate carry or borrow out of bit 3.
F0	PSW.5	Flag 0
		Set/cleared/tested by software as a user-defined status flag.
RS1	PSW.4	Register bank Select control bits 1 & 0.
		Set/cleared by software to determine working register bank (see Note).
RS	PSW.3	

Figure 4. PSW—Program Status Word Organization

A significant side benefit of an instruction set more powerful than those of previous single-chip microcomputers is that it is easier to generate applications-oriented software. Generalized addressing modes for byte and bit instructions reduce the number of source code lines written and debugged for a given application. This leads in turn to proportionately lower software costs, greater reliability, and faster design cycles.

Accumulator and PSW

The 8051, like its 8048 predecessor, is primarily an accumulator-based architecture: an eight-bit register called the accumulator ("A") holds a source operand and receives the result of the arithmetic instructions (addition, subtraction, multiplication, and division). The accumulator can be the source or destination for logical operations and a number of special data movement instructions, including table look-ups and external RAM expansion. Several functions apply exclusively to the accumulator: rotates, parity computation, testing for zero, and so on. Many instructions implicitly or explicitly affect (or are affected by) several status flags, which are grouped together to form the Program Status Word shown in Figure 4.

(The period within entries under the Position column is called the "dot operator," and indicates a particular bit position within an eight-bit byte. "PSW.5" specifies bit 5 of the PSW. Both the documentation and ASM51 use this notation.)

The most "active" status bit is called the carry flag (abbreviated "C"). This bit makes possible multiple precision arithmetic operations including addition, subtraction,

Symbol Position Name and Significance

OV	PSW.2	Overflow flag.
		Set/cleared by hardware during arithmetic instructions to indicate overflow conditions.
—	PSW.1	(reserved)
P	PSW.0	Parity flag.
		Set/cleared by hardware each instruction cycle to indicate an odd/even number of "one" bits in the accumulator, i.e., even parity.
Note—		the contents of (RS1, RS0) enable the working register banks as follows:
	(0,0)—Bank 0	(00H-07H)
	(0,1)—Bank 1	(08H-0FH)
	(1,0)—Bank 2	(10H-17H)
	(1,1)—Bank 3	(18H-1FH)

and rotates. The carry also serves as a "Boolean accumulator" for one-bit logical operations and bit manipulation instructions. The overflow flag (OV) detects when arithmetic overflow occurs on signed integer operands, making two's complement arithmetic possible. The parity flag (P) is updated after every instruction cycle with the even-parity of the accumulator contents.

The CPU does not control the two register-bank select bits, RS1 and RS0. Rather, they are manipulated by software to enable one of the four register banks. The usage of the PSW flags is demonstrated in the Instruction Set chapter of this Note.

Even though the architecture is accumulator-based, provisions have been made to bypass the accumulator in common instruction situations. Data may be moved from any location on-chip to any register, address, or indirect address (and vice versa), any register may be loaded with a constant, etc., all without affecting the accumulator. Logical operations may be performed against registers or variables to alter fields of bits—without using or affecting the accumulator. Variables may be incremented, decremented, or tested without using the accumulator. Flags and control bits may be manipulated and tested without affecting anything else.

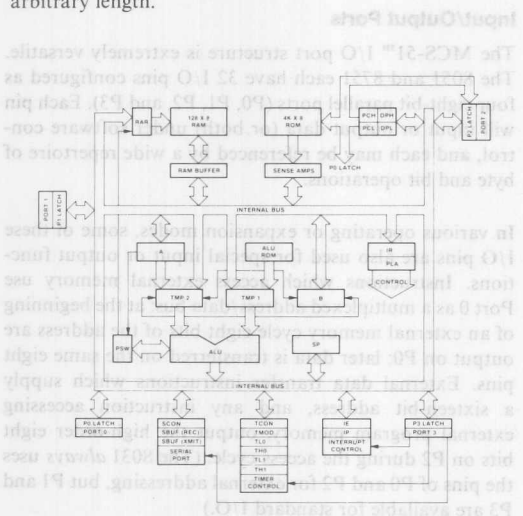
Other CPU Registers

A special eight-bit register ("B") serves in the execution of the multiply and divide instructions. This register is used in conjunction with the accumulator as the second input operand and to return eight-bits of the result.

The MCS-51 family processors include a hardware stack within internal RAM, useful for subroutine linkage,

passing parameters between routines, temporary variable storage, or saving status during interrupt service routines. The Stack Pointer (SP) is an eight-bit pointer register which indicates the address of the last byte pushed onto the stack. The stack pointer is automatically incremented or decremented on all push or pop instructions and all subroutine calls and returns. In theory, the stack in the 8051 may be up to a full 128 bytes deep. (In practice, even simple programs would use a handful of RAM locations for pointers, variables, and so forth—reducing the stack depth by that number.) The stack pointer defaults to 7 on reset, so that the stack will start growing up from location 8, just like in the 8048. By altering the pointer contents the stack may be relocated anywhere within internal RAM.

Finally, a 16-bit register called the data pointer (DPTR) serves as a base register in indirect jumps, table look-up instructions, and external data transfers. The high- and low-order halves of the data pointer may be manipulated as separate registers (DPH and DPL, respectively) or together using special instructions to load or increment all sixteen bits. Unlike the 8048, look-up tables can therefore start anywhere in program memory and be of arbitrary length.



Memory Spaces

Program memory is separate and distinct from data memory. Each memory type has a different addressing mechanism, different control signals, and a different function.

The program memory array (ROM or EPROM), like an elephant, is extremely large and never forgets information, even when power is removed. Program memory is used for information needed each time power is applied: initialization values, calibration constants, keyboard layout tables, etc., as well as the program itself. The program memory has a sixteen-bit address bus; its elements

are addressed using the Program Counter or instructions which generate a sixteen-bit address.

To stretch our analogy just a bit, data memory is like a mouse: it is smaller and therefore quicker than program memory, and it goes into a random state when electrical power is applied. On-chip data RAM is used for variables which are determined or may change while the program is running.

A computer spends most of its time manipulating variables, not constants, and a relatively small number of variables at that. Since eight-bits is more than sufficient to uniquely address 128 RAM locations, the on-chip RAM address register is only one byte wide. In contrast to the program memory, data memory accesses need a single eight-bit value—a constant or another variable—to specify a unique location. Since this is the basic width of the ALU and the different memory types, those resources can be used by the addressing mechanisms, contributing greatly to the computer's operating efficiency.

The partitioning of program and data memory is extended to off-chip memory expansion. Each may be added independently, and each uses the same address and data busses, but with different control signals. External program memory is gated onto the external data bus by the $\overline{\text{PSEN}}$ (Program Store Enable) control output, pin 29. External data memory is read onto the bus by the $\overline{\text{RD}}$ output, pin 17, and written with data supplied from the microcomputer by the $\overline{\text{WR}}$ output, pin 16. (There is no control pin to write external program ROM, which is by definition Read Only.) While both types may be expanded to up to 64K bytes, the external data memory may optionally be expanded in 256 byte "pages" to preserve the use of P2 as an I/O port. This is useful with a relatively small expansion RAM (such as the Intel® 8155) or for addressing external peripherals.

Single-chip controller programs are finalized during the project design cycle, and are not modified after production. Intel's single-chip microcomputers are not "von Neumann" architectures common among main-frame and mini-computer systems: the MCS-51™ processor data memory—on-chip and external—may not be used for program code. Just as there is no write-control signal for program memory, there is no way for the CPU to execute instructions out of RAM. In return, this concession allows an architecture optimized for efficient controller applications: a large, fixed program located in ROM, a hundred or so variables in RAM, and different methods for efficiently addressing each.

(Von Neumann machines are helpful for software development and debug. An 8051 system could be modified to have a single off-chip memory space by gating together the two memory-read controls ($\overline{\text{PSEN}}$ and $\overline{\text{RD}}$) with a two-input AND gate (Figure 5). The CPU could then write data into the common memory array using $\overline{\text{WR}}$ and

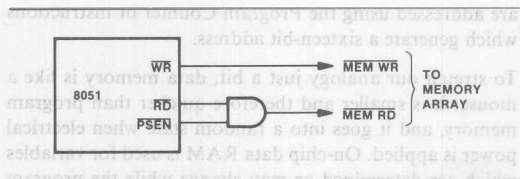


Figure 5. Combining External Program and Data Memory Arrays

external data transfer instructions, and read instructions or data with the AND gate output and data transfer or program memory look-up instructions.)

In addition to the memory arrays, there is (yet) another (albeit sparsely populated) physical address space. Connected to the internal data bus are a score of special-purpose eight-bit registers scattered throughout the chip. Some of these—B, SP, PSW, DPH, and DPL—have been discussed above. Others—I/O ports and peripheral function registers—will be introduced in the following sections. Collectively, these registers are designated as the “special-function register” address space. Even the accumulator is assigned a spot in the special-function register address space for additional flexibility and uniformity.

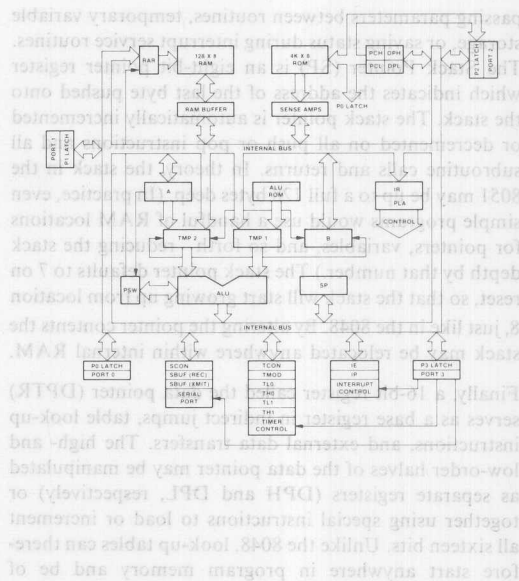
Thus, the MCS-51™ architecture supports several distinct “physical” address spaces, functionally separated at the hardware level by different addressing mechanisms, read and write control signals, or both:

- On-chip program memory;
- On-chip data memory;
- Off-chip program memory;
- Off-chip data memory;
- On-chip special-function registers.

What the *programmer sees*, though, are “logical” address spaces. For example, as far as the programmer is concerned, there is only one type of program memory, 64K bytes in length. The fact that it is formed by combining on- and off-chip arrays (split 4K/60K on the 8051 and 8751) is “invisible” to the programmer; the CPU automatically fetches each byte from the appropriate array, based on its address.

(Presumably, future microcomputers based on the MCS-51™ architecture may have a different physical split, with more or less of the 64K total implemented on-chip. Using the MCS-48™ family as a precedent, the 8048’s 4K potential program address space was split 1K/3K between on- and off-chip arrays; the 8049’s was split 2K/2K.)

Why go into such tedious details about address spaces? The logical addressing modes are described in the Instruction Set chapter in terms of physical address spaces. Understanding their differences now will pay off in understanding and using the chips later.



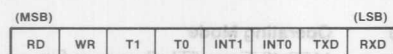
Input/Output Ports

The MCS-51™ I/O port structure is extremely versatile. The 8051 and 8751 each have 32 I/O pins configured as four eight-bit parallel ports (P0, P1, P2, and P3). Each pin will input or output data (or both) under software control, and each may be referenced by a wide repertoire of byte and bit operations.

In various operating or expansion modes, some of these I/O pins are also used for special input or output functions. Instructions which access external memory use Port 0 as a multiplexed address/data bus: at the beginning of an external memory cycle eight bits of the address are output on P0; later data is transferred on the same eight pins. External data transfer instructions which supply a sixteen-bit address, and any instruction accessing external program memory, output the high-order eight bits on P2 during the access cycle. (The 8031 *always* uses the pins of P0 and P2 for external addressing, but P1 and P3 are available for standard I/O.)

The eight pins of Port 3 (P3) each have a special function. Two external interrupts, two counter inputs, two serial data lines, and two timing control strobes use pins of P3 as described in Figure 6. Port 3 pins corresponding to functions not used are available for conventional I/O.

Even within a single port, I/O functions may be combined in many ways: input and output may be performed using different pins at the same time, or the same pins at different times; in parallel in some cases, and in serial in others; as test pins, or (in the case of Port 3) as additional special functions.



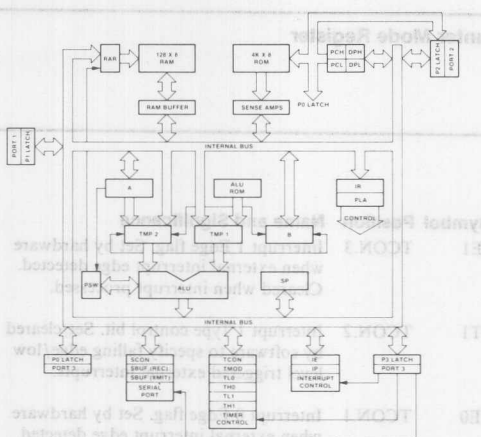
Symbol Position Name and Significance

RD	P3.7	Read data control output. Active low pulse generated by hardware when external data memory is read.
WR	P3.6	Write data control output. Active low pulse generated by hardware when external data memory is written.
T1	P3.5	Timer/counter 1 external input or test pin.
T0	P3.4	Timer/counter 0 external input or test pin.

Symbol Position Name and Significance

INT1	P3.3	Interrupt 1 input pin. Low-level or falling-edge triggered.
INT0	P3.2	Interrupt 0 input pin. Low-level or falling-edge triggered.
TXD	P3.1	Transmit Data pin for serial port in UART mode. Clock output in shift register mode.
RXD	P3.0	Receive Data pin for serial port in UART mode. Data I/O pin in shift register mode.

Figure 6. P3—Alternate Special Functions of Port 3



Special Peripheral Functions

There are a few special needs common among control-oriented computer systems:

- keeping track of elapsed real-time;
- maintaining a count of signal transitions;
- measuring the precise width of input pulses;
- communicating with other systems or people;
- closely monitoring asynchronous external events.

Until now, microprocessor systems needed peripheral chips such as timer/counters, USARTs, or interrupt controllers to meet these needs. The 8051 integrates all of these capabilities on-chip!

Timer/Counters

There are two sixteen-bit multiple-mode Timer/Counters on the 8051, each consisting of a "High" byte (corresponding to the 8048 "T" register) and a low byte (similar to the 8048 prescaler, with the additional flexibility of being

software-accessible). These registers are called, naturally enough, TH0, TL0, TH1, and TL1. Each pair may be independently software programmed to any of a dozen modes with a mode register designated TMOD (Figure 7), and controlled with register TCON (Figure 8).

The timer modes can be used to measure time intervals, determine pulse widths, or initiate events, with one-micro-second resolution, up to a maximum interval of 65,536 instruction cycles (over 65 milliseconds). Longer delays may easily be accumulated through software. Configured as a counter, the same hardware will accumulate external events at frequencies from D.C. to 500 KHz, with up to sixteen bits of precision.

Serial Port Interface

Each microcomputer contains a high-speed, full-duplex, serial port which is software programmable to function in four basic modes: shift-register I/O expander, 8-bit UART, 9-bit UART, or interprocessor communications link. The UART modes will interface with standard I/O devices (e.g. CRTs, teletypewriters, or modems) at data rates from 122 baud to 31 kilobaud. Replacing the standard 12 MHz crystal with a 10.7 MHz crystal allows 110 baud. Even or odd parity (if desired) can be included with simple bit-handling software routines. Inter-processor communications in distributed systems takes place at 187 kilobaud with hardware for automatic address/data message recognition. Simple TTL or CMOS shift registers provide low-cost I/O expansion at a super-fast 1 Megabaud. The serial port operating modes are controlled by the contents of register SCON (Figure 9).

Interrupt Capability and Control

(Interrupt capability is generally considered a CPU function. It is being introduced here since, from an applications point of view, interrupts relate more closely to peripheral and system interfacing.)

(MSB)				(LSB)			
GATE	C/T	M1	M0	GATE	C/T	M1	M0
TIMER 1				TIMER 0			

GATE	Gating control. When set, Timer/counter "x" is enabled only while "INTx" pin is high and "TRx" control bit is set. When cleared, timer/counter is enabled whenever "TRx" control bit is set.
C/T	Timer or Counter Selector. Cleared for Timer operation (input from internal system clock). Set for Counter operation (input from "Tx" input pin).

M1	M0	Operating Mode
0	0	MCS-48 Timer. "TLx" serves as five-bit prescaler.
0	1	16-bit timer/counter. "THx" and "TLx" are cascaded; there is no prescaler.
1	0	8-bit auto-reload timer/counter. "THx" holds a value which is to be reloaded into "TLx" each time it overflows.
1	1	(Timer 0) TL0 is an eight-bit timer/counter controlled by the standard Timer 0 control bits. TH0 is an eight-bit timer only controlled by Timer 1 control bits.
1	1	(Timer 1) Timer/counter 1 stopped.

Figure 7. TMOD—Timer/Counter Mode Register

(MSB)				(LSB)			
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
<p>Symbol Position Name and Significance</p> <p>TF1 TCON.7 Timer 1 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.</p> <p>TR1 TCON.6 Timer 1 Run control bit. Set/cleared by software to turn timer/counter on/off.</p> <p>TF0 TCON.5 Timer 0 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.</p> <p>TR0 TCON.4 Timer 0 Run control bit. Set/cleared by software to turn timer/counter on/off.</p>							
<p>IE1 TCON.3 Interrupt 1 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.</p> <p>IT1 TCON.2 Interrupt 1 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.</p> <p>IE0 TCON.1 Interrupt 0 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.</p> <p>IT0 TCON.0 Interrupt 0 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.</p>							

Figure 8. TCON—Timer/Counter Control/Status Register

(MSB)								(LSB)							
SM0	SM1	SM2	REN	TB8	RB8	TI	RI								
Symbol	Position	Name and Significance													
SM0	SCON.7	Serial port Mode control bit 0. Set/cleared by software (see note).													
SM1	SCON.6	Serial port Mode control bit 1. Set/cleared by software (see note).													
SM2	SCON.5	Serial port Mode control bit 2. Set by software to disable reception of frames for which bit 8 is zero.													
REN	SCON.4	Receiver Enable control bit. Set/cleared by software to enable/disable serial data reception.													
TB8	SCON.3	Transmit Bit 8. Set/cleared by hardware to determine state of ninth data bit transmitted in 9-bit UART mode.													
		Note— the state of (SM0,SM1) selects: (0,0)—Shift register I/O expansion. (0,1)—8 bit UART, variable data rate. (1,0)—9 bit UART, fixed data rate. (1,1)—9 bit UART, variable data rate.													

Figure 9. SCON—Serial Port Control/Status Register

These peripheral functions allow special hardware to monitor real-time signal interfacing without bothering the CPU. For example, imagine serial data is arriving from one CRT while being transmitted to another, and one timer/counter is tallying high-speed input transitions while the other measures input pulse widths. During all of this the CPU is thinking about something else.

But how does the CPU know when a reception, transmission, count, or pulse is finished? The 8051 programmer can choose from three approaches.

TCON and SCON contain status bits set by the hardware when a timer overflows or a serial port operation is completed. The first technique reads the control register into the accumulator, tests the appropriate bit, and does a conditional branch based on the result. This "polling" scheme (typically a three-instruction sequence though additional instructions to save and restore the accumulator may sometimes be needed) will surely be familiar to programmers used to multi-chip microcomputer systems and peripheral controller chips. This process is rather cumbersome, especially when monitoring multiple peripherals.

As a second approach, the 8051 can perform a conditional branch based on the state of any control or status bit or input pin in a single instruction; a four instruction sequence could poll the four simultaneous happenings mentioned above in just eight microseconds.

Unfortunately, the CPU must still drop what it's doing to test these bits. A manager cannot do his own work well if he is continuously monitoring his subordinates; they should interrupt him (or her) only when they need attention or guidance. So it is with machines: ideally, the CPU would not have to worry about the peripherals until they require servicing. At that time, it would postpone the

background task long enough to handle the appropriate device, then return to the point where it left off.

This is the basis of the third and generally optimal solution, hardware interrupts. The 8051 has five interrupt sources: one from the serial port when a transmission or reception is complete, two from the timers when overflows occur, and two from input pins INT0 and INT1. Each source may be independently enabled or disabled to allow polling on some sources or at some times, and each may be classified as high or low priority. A high priority source can interrupt a low priority service routine; the manager's boss can interrupt conferences with subordinates. These options are selected by the interrupt enable and priority control registers, IE and IP (Figures 10 and 11).

Each source has a particular program memory address associated with it (Table 3), starting at 0003H (as in the 8048) and continuing at eight-byte intervals. When an event enabled for interrupts occurs the CPU automatically executes an internal subroutine call to the corresponding address. A user subroutine starting at this location (or jumped to from this location) then performs the instructions to service that particular source. After completing the interrupt service routine, execution returns to the background program.

Table 3. 8051 Interrupt Sources and Service Vectors

Interrupt Source	Service Routine Starting Address
(Reset)	0000H
External 0	0003H
Timer/Counter 0	000BH
External 1	0013H
Timer/Counter 1	001BH
Serial Port	0023H

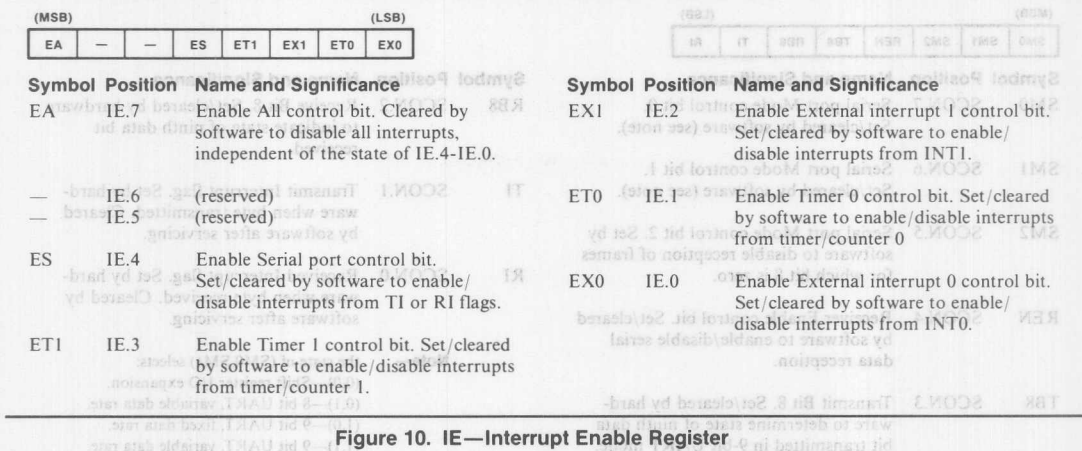


Figure 10. IE—Interrupt Enable Register

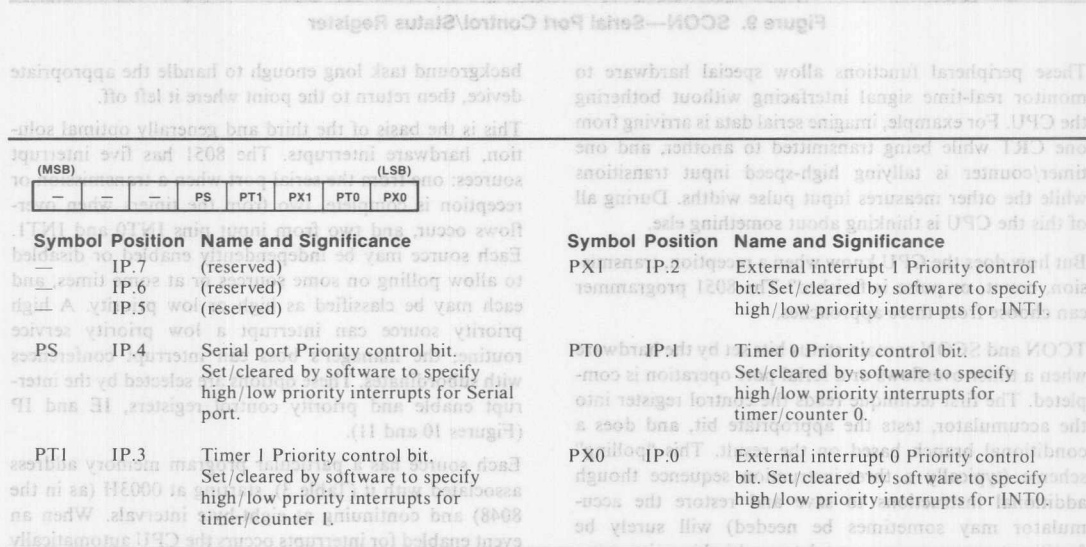


Figure 11. IP—Interrupt Priority Control Register

Table 3. 8051 Interrupt Sources and Service Vectors

Interrupt Source	Service Routine Starting Address
Serial Port	0023H
Timer Counter 1	001BH
External 1	0013H
Timer Counter 0	000BH
External 0	0007H
(Reset)	0000H

Table 4. MCS-51™ Instruction Set Description

ARITHMETIC OPERATIONS				DATA TRANSFER (cont.)			
Mnemonic	Description	Byte	Cyc	Mnemonic	Description	Byte	Cyc
ADD A,Rn	Add register to Accumulator	1	1	MOV A,@A+DPTR	Move Code byte relative to DPTR to A	1	2
ADD A,direct	Add direct byte to Accumulator	2	1	MOV A,@A+PC	Move Code byte relative to PC to A	1	2
ADD A,@Ri	Add indirect RAM to Accumulator	1	1	MOVX A,@Ri	Move External RAM (8-bit addr) to A	1	2
ADD A,#data	Add immediate data to Accumulator	2	1	MOVX A,@DPTR	Move External RAM (16-bit addr) to A	1	2
ADDC A,Rn	Add register to Accumulator with Carry	1	1	MOVX @Ri,A	Move A to External RAM (8-bit addr)	1	2
ADDC A,direct	Add direct byte to A with Carry flag	2	1	MOVX @DPTR,A	Move A to External RAM (16-bit addr)	1	2
ADDC A,@Ri	Add indirect RAM to A with Carry flag	1	1	PUSH direct	Push direct byte onto stack	2	2
ADDC A,#data	Add immediate data to A with Carry flag	2	1	POP direct	Pop direct byte from stack	2	2
SUBB A,Rn	Subtract register from A with Borrow	1	1	XCH A,Rn	Exchange register with Accumulator	1	1
SUBB A,direct	Subtract direct byte from A with Borrow	2	1	XCH A,direct	Exchange direct byte with Accumulator	2	1
SUBB A,@Ri	Subtract indirect RAM from A w/ Borrow	1	1	XCH A,@Ri	Exchange indirect RAM with A	1	1
SUBB A,#data	Subtract immed. data from A w/ Borrow	2	1	XCHD A,@Ri	Exchange low-order Digit ind. RAM w/A	1	1
INC A	Increment Accumulator	1	1	BOOLEAN VARIABLE MANIPULATION			
INC Rn	Increment register	1	1	Mnemonic	Description	Byte	Cyc
INC direct	Increment direct byte	2	1	CLR C	Clear Carry flag	1	1
INC @Ri	Increment indirect RAM	1	1	CLR bit	Clear direct bit	2	1
DEC A	Decrement Accumulator	1	1	SETB C	Set Carry flag	1	1
DEC Rn	Decrement register	1	1	SETB bit	Set direct Bit	2	1
DEC direct	Decrement direct byte	2	1	CPL C	Complement Carry flag	1	1
DEC @Ri	Decrement indirect RAM	1	1	CPL bit	Complement direct bit	2	1
INC DPTR	Increment Data Pointer	1	2	ANL C,bit	AND direct bit to Carry flag	2	2
MUL AB	Multiply A & B	1	4	ANL C,bit	AND complement of direct bit to Carry	2	2
DIV AB	Divide A by B	1	4	ORL C,bit	OR direct bit to Carry flag	2	2
DA A	Decimal Adjust Accumulator	1	1	ORL C,bit	OR complement of direct bit to Carry	2	2
LOGICAL OPERATIONS				MOV C,bit	Move direct bit to Carry flag	2	1
Mnemonic	Description	Byte	Cyc	MOV bit,C	Move Carry flag to direct bit	2	2
ANL A,Rn	AND register to Accumulator	1	1	PROGRAM AND MACHINE CONTROL			
ANL A,direct	AND direct byte to Accumulator	2	1	Mnemonic	Description	Byte	Cyc
ANL A,@Ri	AND indirect RAM to Accumulator	1	1	ACALL addr11	Absolute Subroutine Call	2	2
ANL A,#data	AND immediate data to Accumulator	2	1	LCALL addr16	Long Subroutine Call	3	2
ANL direct,A	AND Accumulator to direct byte	2	1	RET	Return from subroutine	1	2
ANL direct,#data	AND immediate data to direct byte	3	2	RETI	Return from interrupt	1	2
ORL A,Rn	OR register to Accumulator	1	1	AJMP addr11	Absolute Jump	2	2
ORL A,direct	OR direct byte to Accumulator	2	1	LJMP addr16	Long Jump	3	2
ORL A,@Ri	OR indirect RAM to Accumulator	1	1	SJMP rel	Short Jump (relative to the DPTR)	2	2
ORL A,#data	OR immediate data to Accumulator	2	1	JMP @A+DPTR	Jump indirect relative to the DPTR	1	2
ORL direct,A	OR Accumulator to direct byte	2	1	JZ rel	Jump if Accumulator is Zero	2	2
ORL direct,#data	OR immediate data to direct byte	3	2	JNZ rel	Jump if Accumulator is Not Zero	2	2
XRL A,Rn	Exclusive-OR register to Accumulator	1	1	JC rel	Jump if Carry flag is set	2	2
XRL A,direct	Exclusive-OR direct byte to Accumulator	2	1	JNC rel	Jump if No Carry flag	2	2
XRL A,@Ri	Exclusive-OR indirect RAM to A	1	1	JB bit,rel	Jump if direct Bit set	3	2
XRL A,#data	Exclusive-OR immediate data to A	2	1	JNB bit,rel	Jump if direct Bit Not set	3	2
XRL direct,A	Exclusive-OR Accumulator to direct byte	2	1	JBC bit,rel	Jump if direct Bit is set & Clear bit	3	2
XRL direct,#data	Exclusive-OR immediate data to direct	3	2	CJNE A,direct,rel	Compare direct to A & Jump if Not Equal	3	2
CLR A	Clear Accumulator	1	1	CJNE A,#data,rel	Comp. immed. to A & Jump if Not Equal	3	2
CPL A	Complement Accumulator	1	1	CJNE Rn,#data,rel	Comp. immed. to reg. & Jump if Not Equal	3	2
RL A	Rotate Accumulator Left	1	1	@Ri,#data,rel	Comp. immed. to ind. & Jump if Not Equal	3	2
RLC A	Rotate A Left through the Carry flag	1	1	Rn,rel	Decrement register & Jump if Not Zero	2	2
RR A	Rotate Accumulator Right	1	1	DJNZ direct,rel	Decrement direct & Jump if Not Zero	3	2
RRC A	Rotate A Right through Carry flag	1	1	NOP	No operation	1	1
SWAP A	Swap nibbles within the Accumulator	1	1	Notes on data addressing modes:			
DATA TRANSFER				Rn	Working register R0-R7		
Mnemonic	Description	Byte	Cyc	direct	128 internal RAM locations, any I/O port, control or status register		
MOV A,Rn	Move register to Accumulator	1	1	@Ri	Indirect internal RAM location addressed by register R0 or R1		
MOV A,direct	Move direct byte to Accumulator	2	1	#data	8-bit constant included in instruction		
MOV A,@Ri	Move indirect RAM to Accumulator	1	1	#data16	16-bit constant included as bytes 2 & 3 of instruction		
MOV A,#data	Move immediate data to Accumulator	2	1	bit	128 software flags, any I/O pin, control or status bit		
MOV Rn,A	Move Accumulator to register	1	1	Notes on program addressing modes:			
MOV Rn,direct	Move direct byte to register	2	2	addr16	Destination address for LCALL & LJMP may be anywhere within the 64-Kilobyte program memory address space.		
MOV Rn,#data	Move immediate data to register	2	1	addr11	Destination address for ACALL & AJMP will be within the same 2-Kilobyte page of program memory as the first byte of the following instruction.		
MOV direct,A	Move Accumulator to direct byte	2	1	rel	SJMP and all conditional jumps include an 8-bit offset byte. Range is +127/-128 bytes relative to first byte of the following instruction.		
MOV direct,Rn	Move register to direct byte	2	2	All mnemonics copyrighted © Intel Corporation 1979			
MOV direct,direct	Move direct byte to direct	3	2				
MOV direct,@Ri	Move indirect RAM to direct byte	2	2				
MOV direct,#data	Move immediate data to direct byte	3	2				
MOV @Ri,A	Move Accumulator to indirect RAM	1	1				
MOV @Ri,direct	Move direct byte to indirect RAM	2	2				
MOV @Ri,#data	Move immediate data to indirect RAM	2	1				
MOV DPTR,#data16	Load Data Pointer with a 16-bit constant	3	2				

3. INSTRUCTION SET AND ADDRESSING MODES

The 8051 instruction set is extremely regular, in the sense that most instructions can operate with variables from several different physical or logical address spaces. Before getting deeply enmeshed in the instruction set proper, it is important to understand the details of the most common data addressing modes. Whereas Table 4 summarizes the instructions set broken down by functional

group, this chapter starts with the addressing mode classes and builds to include the related instructions.

Data Addressing Modes

MCS-51 assembly language instructions consist of an operation mnemonic and zero to three operands separated by commas. In two operand instructions the destination is specified first, then the source. Many byte-wide data

operations (such as ADD or MOV) inherently use the accumulator as a source operand and/or to receive the result. For the sake of clarity the letter "A" is specified in the source or destination field in all such instructions. For example, the instruction,

```
ADD A,<source>
```

will add the variable<source>to the accumulator, leaving the sum in the accumulator.

The operand designated "<source>" above may use any of four common logical addressing modes:

- Register—one of the working registers in the currently enabled bank.
- Direct—an internal RAM location, I/O port, or special-function register.
- Register-indirect—an internal RAM location, pointed to by a working register.
- Immediate data—an eight-bit constant incorporated into the instruction.

The first three modes provide access to the internal RAM and Hardware Register address spaces, and may therefore be used as source or destination operands; the last mode accesses program memory and may be a source operand only.

(It is hard to show a "typical application" of any instruction without involving instructions not yet described. The following descriptions use only the self-explanatory ADD and MOV instructions to demonstrate how the four addressing modes are specified and used. Subsequent examples will become increasingly complex.)

Register Addressing

The 8051 programmer has access to eight "working registers," numbered R0-R7. The least-significant three-bits of the instruction opcode indicate one register within this logical address space. Thus, a function code and operand address can be combined to form a short (one byte) instruction (Figure 12.a).

The 8051 assembly language indicates register addressing with the symbol Rn (where n is from 0 to 7) or with a symbolic name previously defined as a register by the EQUate or SET directives. (For more information on assembler directives see the Macro Assembler Reference Manual.)

Example 1—Adding Two Registers Together

```
REGADR ADD CONTENTS OF REGISTER 1
        TO CONTENTS OF REGISTER 0
REGADR: MOV A,R0
        ADD A,R1
        MOV RO,A
```

There are four such banks of working registers, only one of which is active at a time. Physically, they occupy the first 32 bytes of on-chip data RAM (addresses 0-1FH). PSW bits 4 and 3 determine which bank is active. A

hardware reset enables register bank 0; to select a different bank the programmer modifies PSW bits 4 and 3 accordingly.

Example 2—Selecting Alternate Memory Banks

```
MOV PSW,#00010000B ;SELECT BANK 2
```

Register addressing in the 8051 is the same as in the 8048 family, with two enhancements: there are four banks rather than one or two, and 16 instructions (rather than 12) can access them.

Direct Byte Addressing

Direct addressing can access any on-chip variable or hardware register. An additional byte appended to the opcode specifies the location to be used (Figure 12.b).

Depending on the highest order bit of the direct address byte, one of two physical memory spaces is selected. When the direct address is between 0 and 127 (00H-7FH) one of the 128 low-order on-chip RAM locations is used. (Future microcomputers based on the MCS-51™ architecture may incorporate more than 128 bytes of on-chip RAM. Even if this is the case, only the low-order 128 bytes will be directly addressable. The remainder would be accessed indirectly or via the stack pointer.)

Example 3—Adding RAM Location Contents

```
;DIRADR ADD CONTENTS OF RAM LOCATION 41H
        TO CONTENTS OF RAM LOCATION 40H
DIRADR: MOV A,40H
        ADD A,41H
        MOV 40H,A
```

All I/O ports and special function, control, or status registers are assigned addresses between 128 and 255 (80H-OFFH). When the direct address byte is between these limits the corresponding hardware register is accessed. For example, Ports 0 and 1 are assigned direct addresses 80H and 90H, respectively. A complete list is presented in Table 5. Don't waste your time trying to memorize the addresses in Table 5. Since programs using absolute addresses for function registers would be difficult to write or understand, ASM51 allows and understands the abbreviations listed instead.

Example 4—Adding Input Port Data to Output Port Data

```
;PRTADR ADD DATA INPUT ON PORT 1
        TO DATA PREVIOUSLY OUTPUT
        ON PORT 0
PRTADR: MOV A,P0
        ADD A,P1
        MOV P0,A
```

Direct addressing allows all special-function registers in the 8051 to be read, written, or used as instruction operands. In general, this is the *only* method used for accessing I/O ports and special-function registers. If direct addressing is used with special-function register addresses other than those listed, the result of the instruction is undefined.

The 8048 does not have or need any generalized direct addressing mode, since there are only five special registers (BUS, P1, P2, PSW, & T) rather than twenty. Instead, 16 special 8048 opcodes control output bits or read or write each register to the accumulator. These functions are all subsumed by four of the 27 direct addressing instructions of the 8051.

Table 5. 8051 Hardware Register Direct Addresses

Register	Address	Function
P0	80H*	Port 0
SP	81H	Stack Pointer
DPL	82H	Data Pointer (Low)
DPH	83H	Data Pointer (High)
TCON	88H*	Timer register
TMOD	89H	Timer Mode register
TL0	8AH	Timer 0 Low byte
TL1	8BH	Timer 1 Low byte
TH0	8CH	Timer 0 High byte
TH1	8DH	Timer 1 High byte
P1	90H*	Port 1
SCON	98H*	Serial Port Control register
SBUF	99H	Serial Port data Buffer
P2	0A0H*	Port 2
IE	0A8H*	Interrupt Enable register
P3	0B0H*	Port 3
IP	0B8H*	Interrupt Priority register
PSW	0D0H*	Program Status Word
ACC	0E0H*	Accumulator (direct address)
B	0F0H*	B register

* = bit addressable register.

Register-Indirect Addressing

How can you handle variables whose locations in RAM are determined, computed, or modified while the program is running? This situation arises when manipulating sequential memory locations, indexed entries within tables in RAM, and multiple precision or string operations. Register or Direct addressing cannot be used, since their operand addresses are fixed at assembly time.

The 8051 solution is "register-indirect RAM addressing." R0 and R1 of each register bank may operate as index or pointer registers, their contents indicating an address into RAM. The internal RAM location so addressed is the actual operand used. The least significant bit of the instruction opcode determines which register is used as the "pointer" (Figure 12.c).

In the 8051 assembly language, register-indirect addressing is represented by a commercial "at" sign ("@") preceding R0, R1, or a symbol defined by the user to be equal to R0 or R1.

Example 5—Indirect Addressing

```

INDADR ADD CONTENTS OF MEMORY LOCATION
        ADDRESSED BY REGISTER 1
        TO CONTENTS OF RAM LOCATION
        ADDRESSED BY REGISTER 0
INDADR MOV A, @R0
ADD A, @R1
MOV @R0, A

```

Indirect addressing on the 8051 is the same as in the 8048 family, except that all eight bits of the pointer register contents are significant; if the contents point to a non-existent memory location (i.e., an address greater than 7FH on the 8051) the result of the instruction is undefined. (Future microcomputers based on the MCS-51™ architecture could implement additional memory in the on-chip RAM logical address space at locations above 7FH.) The 8051 uses register-indirect addressing for five new instructions plus the 13 on the 8048.

Immediate Addressing

When a source operand is a constant rather than a variable (i.e., the instruction uses a value known at assembly time), then the constant can be incorporated into the instruction. An additional instruction byte specifies the value used (Figure 12.d).

The value used is fixed at the time of ROM manufacture or EPROM programming and may not be altered during program execution. In the assembly language immediate operands are preceded by a number sign ("#"). The operand may be either a numeric string, a symbolic variable, or an arithmetic expression using constants.

Example 6—Adding Constants Using Immediate Addressing

```

IMMADR ADD THE CONSTANT 12 (DECIMAL)
        TO THE CONSTANT 34 (DECIMAL)
        LEAVE SUM IN ACCUMULATOR
IMMADR MOV A, #12
ADD A, #34

```

The preceding example was included for consistency; it has little practical value. Instead, ASM51 could compute the sum of two constants at assembly time.

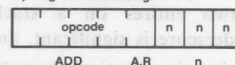
Example 7—Adding Constants Using ASM51 Capabilities

```

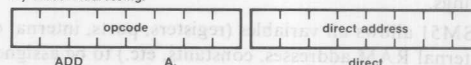
ASM51 LOAD ACC WITH THE SUM OF
        THE CONSTANT 12 (DECIMAL) AND
        THE CONSTANT 34 (DECIMAL)
ASM51 MOV A, #(12+34)

```

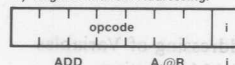
a.) Register Addressing:



b.) Direct Addressing:



c.) Register-Indirect Addressing:



d.) Immediate Addressing:

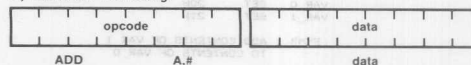


Figure 12. Data Addressing Machine Code Formats

Addressing Mode Combinations

The above examples all demonstrated the use of the four data-addressing modes in two-operand instructions (MOV, ADD) which use the accumulator as one operand. The operations ADDC, SUBB, ANL, ORL, and XRL (all to be discussed later) could be substituted for ADD in each example. The first three modes may be also be used for the XCH operation or, in combination with the Immediate Addressing mode (and an additional byte), loaded with a constant. The one-operand instructions INC and DEC, DJNZ, and CJNE may all operate on the accumulator, or may specify the Register, Direct, and Register-indirect addressing modes. Exception: as in the 8048, DJNZ cannot use the accumulator or indirect addressing. (The PUSH and POP operations cannot inherently address the accumulator as a special register either. However, all three can *directly* address the accumulator as one of the twenty special-function registers by putting the symbol "ACC" in the operand field.)

Advantages of Symbolic Addressing

Like most assembly or higher-level programming languages, ASM51 allows instructions or variables to be given appropriate, user-defined symbolic names. This is done for instruction lines by putting a label followed by a colon (":") before the instruction proper, as in the above examples. Such symbols must start with an alphabetic character (remember what distinguished BACH from 0BACH?), and may include any combination of letters, numbers, question marks ("?",) and underscores ("_"). For very long names only the first 31 characters are relevant.

Assembly language programs may intermix upper- and lower-case letters arbitrarily, but ASM51 converts both to upper-case. For example, ASM51 will internally process an "I" for an "i" and, of course, "A_TOOTH" for "a_tooth."

The underscore character makes symbols easier to read and can eliminate potential ambiguity (as in the label for a subroutine to switch two entires on a stack, "S_EXCHANGE"). The underscore is significant, and would distinguish between otherwise-identical character strings.

ASM51 allows *all* variables (registers, ports, internal or external RAM addresses, constants, etc.) to be assigned labels according to these rules with the EQUate or SET directives.

Example 8—Symbolic Addressing of Variables Defined as RAM Locations

```

VAR_0 SET 20H
VAR_1 SET 21H
SYMB_1 ADD CONTENTS OF VAR_1
        TO CONTENTS OF VAR_0
SYMB_1 MOV A, VAR_0
        ADD A, VAR_1
        MOV VAR_0, A

```

Notice from Table 4 that the MCS-51™ instruction set has relatively few instruction mnemonics (abbreviations) for the programmer to memorize. Different data types or addressing modes are determined by the operands specified, rather than variations on the mnemonic. For example, the mnemonic "MOV" is used by 18 different instructions to operate on three data types (bit, byte, and address). The fifteen versions which move byte variables between the logical address spaces are diagrammed in Figure 13. Each arrow shows the direction of transfer from source to destination.

Notice also that for most instructions allowing register addressing there is a corresponding direct addressing instruction and vice versa. This lets the programmer begin writing 8051 programs as if (s)he has access to 128 different registers. When the program has evolved to the point where the programmer has a fairly accurate idea how often each variable is used, he/she may allocate the working registers in each bank to the most "popular" variables. (The assembly cross-reference option will show exactly how often and where each symbol is referenced.) If symbolic addressing is used in writing the source program only the lines containing the symbol definition will need to be changed; the assembler will produce the appropriate instructions even though the rest of the program is left untouched. Editing only the first two lines of Example 8 will shrink the six-byte code segment produced in half.

How are instruction sets "counted"? There is no standard practice; different people assessing the same CPU using different conventions may arrive at different totals.

Each operation is then broken down according to the different addressing modes (or combinations of addressing modes) it can accommodate. The "CLR" mnemonic is used by two instructions with respect to bit variables ("CLR C" and "CLR bit") and once ("CLR A") with regards to bytes. This expansion yields the 111 separate instructions of Table 4.

The method used for the MCS-51® instruction set first breaks it down into "operations": a basic function applied to a single data type. For example, the four versions of the ADD instruction are grouped to form one operation — addition of eight-bit variables. The six forms of the ANL instruction for byte variables make up a different operation; the two forms of ANL which operate on *bits* are considered still another. The MOV mnemonic is used by three different operation classes, depending on whether bit, byte, or 16-bit values are affected. Using this terminology the 8051 can perform 51 different operations.

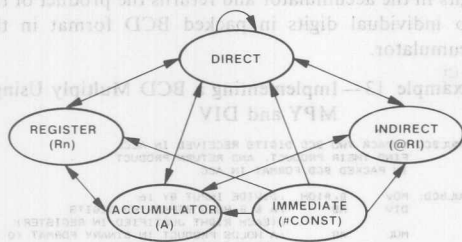


Figure 13. Road map for moving data bytes

Example 9—Redeclaring Example 8 Symbols as Registers

```

VAR_0 SET R0
VAR_1 SET R1
SYMB_2 ADD CONTENTS OF VAR_1
        TO CONTENTS OF VAR_0
SYMB_2: MOV A, VAR_0
        ADD A, VAR_1
        MOV VAR_0, A

```

Arithmetic Instruction Usage — ADD, ADDC, SUBB and DA

The ADD instruction adds a byte variable with the accumulator, leaving the result in the accumulator. The carry flag is set if there is an overflow from bit 7 and cleared otherwise. The AC flag is set to the carry-out from bit 3 for use by the DA instruction described later. ADDC adds the previous contents of the carry flag with the two byte variables, but otherwise is the same as ADD.

The SUBB (subtract with borrow) instruction subtracts the byte variable indicated and the contents of the carry flag together from the accumulator, and puts the result back in the accumulator. The carry flag serves as a "Borrow Required" flag during subtraction operations; when a greater value is subtracted from a lesser value (as in subtracting 5 from 1) requiring a borrow into the highest order bit, the carry flag is set; otherwise it is cleared.

When performing signed binary arithmetic, certain combinations of input variables can produce results which seem to violate the Laws of Mathematics. For example, adding 7FH (127) to itself produces a sum of 0FEH, which is the two's complement representation of -2 (refer back to Table 2)! In "normal" arithmetic, two positive values can't have a negative sum. Similarly, it is normally impossible to subtract a positive value from a negative value and leave a positive result — but in two's complement there are instances where this too may happen. Fundamentally, such anomalies occur when the magnitude of the resulting value is too great to "fit" into the seven bits allowed for it; there is no one-byte two's complement representation for 254, the true sum of 127 and 127.

The MCS-51™ processors detect whether these situations occur and indicate such errors with the OV flag. (OV may be tested with the conditional jump instructions JB and JNB, described under the Boolean Processor chapter.)

At a hardware level, OV is set if there is a carry out of bit 6 but not out of bit 7, or a carry out of bit 7 but not out of bit 6. When adding signed integers this indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands; on SUBB this indicates a negative result after subtracting a negative number from a positive number, or a positive result when a positive number is subtracted from a negative number.

The ADDC and SUBB instructions incorporate the previous state of the carry (borrow) flag to allow multiple precision calculations by repeating the operation with successively higher-order operand bytes. In either case, the carry must be cleared before the first iteration.

If the input data for a multiple precision operation is an unsigned string of integers, upon completion the carry flag will be set if an overflow (for ADDC) or underflow (for SUBB) occurs. With two's complement signed data (i.e., if the most significant bit of the original input data indicates the sign of the string), the overflow flag will be set if overflow or underflow occurred.

Example 10—String Subtraction with Signed Overflow Detection

```

SUBSTR SUBTRACT STRING INDICATED BY R1
        FROM STRING INDICATED BY R0 TO
        PRECISION INDICATED BY R2.
        CHECK FOR SIGNED UNDERFLOW WHEN DONE.

SUBSTR: CLR C          ; BORROW = 0
        MOV A, @R0      ; SUBTRACT NEXT PLACE
        SUBB A, @R1
        MOV @R0, A
        INC R0           ; BUMP POINTERS
        INC R1
        DJNZ R2, SUBS1   ; LOOP AS NEEDED
        WHEN DONE, TEST IF OVERFLOW OCCURRED
        ON LAST ITERATION OF LOOP
        JNB OV, DV_OK
        ; (OVERFLOW RECOVERY ROUTINE)
        RET
OV_OK:  RET

```

Decimal addition is possible by using the DA instruction in conjunction with ADD and/or ADDC. The eight-bit binary value in the accumulator resulting from an earlier addition of two variables (each a packed BCD digit-pair) is adjusted to form two BCD digits of four bits each. If the contents of accumulator bits 3-0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag had been set, six is added to the accumulator producing the proper BCD digit in the low-order nibble. (This addition might itself set — but would not clear — the carry flag.) If the carry flag is set, or if the four high-order bits now exceed nine (1010xxxx-1111xxxx), these bits are incremented by six. The carry flag is left set if originally set or if either addition of six produces a carry out of the highest-order bit, indicating the sum of the original two BCD variables is greater than or equal to decimal 100.

Example 11—Two Byte Decimal Add with Registers and Constants

```
BCDADD ADD THE CONSTANT 1,234 (DECIMAL) TO THE
; CONTENTS OF REGISTER PAIR (R3)-(R2)
; (ALREADY A 4 BCD-DIGIT VARIABLE)

BCDADD MOV A,R2
ADD A,#34H
DA A
MOV R2,A
MOV A,R3
ADDC A,#12H
DA A
MOV R3,A
RET
```

Multiplication and Division

The instruction "MUL AB" multiplies the unsigned eight-bit integer values held in the accumulator and B-register. The low-order byte of the sixteen-bit product is left in the accumulator, the higher-order byte in B. If the high-order eight-bits of the product are all zero the overflow flag is cleared; otherwise it is set. The programmer can poll OV to determine when the B register is non-zero and must be processed.

"DIV AB" divides the unsigned eight-bit integer in the accumulator by the unsigned eight-bit integer in the B-register. The integer part of the quotient is returned in the accumulator; the remainder in the B-register. If the B-register originally contained 00H then the overflow flag will be set to indicate a division error, and the values returned will be undefined. Otherwise OV is cleared. The divide instruction is also useful for purposes such as radix conversion or separating bit fields of the accumulator. A short subroutine can convert an eight-bit unsigned binary integer in the accumulator (between 0 & 255) to a three-digit (two byte) BCD representation. The hundred's digit is returned in one register (HUND) and the ten's and one's digits returned as packed BCD in another (TENONE).

Example 12—Use of DIV Instruction for Radix Conversion

```
; BINBCD CONVERT 8-BIT BINARY VARIABLE IN ACC
; TO 3-DIGIT PACKED BCD FORMAT
; HUNDREDS' PLACE LEFT IN VARIABLE 'HUND',
; TENS' AND ONES' PLACES IN 'TENONE'

HUND EQU 21H
TENONE EQU 22H

BINBCD MOV B,#100 ; DIVIDE BY 100 TO
DIV AB ; DETERMINE NUMBER OF HUNDREDS
MOV HUND,AB
MOV A,B
XCH A,B ; DIVIDE REMAINDER BY 10 TO
DIV AB ; DETERMINE # OF TENS LEFT
XCH A,B ; TENS DIGIT IN ACC, REMAINDER IS ONES
; DIGIT

SHAP ADD A,B ; PACK BCD DIGITS IN ACC
MOV TENONE,A
RET
```

The divide instruction can also separate eight bits of data in the accumulator into sub-fields. For example, packed BCD data may be separated into two nibbles by dividing the data by 16, leaving the high-nibble in the accumulator and the low-order nibble (remainder) in B. The two digits may then be operated on individually or in conjunction with each other. This example receives two packed BCD

digits in the accumulator and returns the product of the two individual digits in packed BCD format in the accumulator.

Example 13—Implementing a BCD Multiply Using MPY and DIV

```
; MULBCD UNPACK TWO BCD DIGITS RECEIVED IN ACC,
; FIND THEIR PRODUCT, AND RETURN PRODUCT
; IN PACKED BCD FORMAT IN ACC

MULBCD MOV B,#10H ; DIVIDE INPUT BY 16
DIV AB ; A & B HOLD SEPARATED DIGITS
; (EACH RIGHT JUSTIFIED IN REGISTER)
MUL AB ; A HOLDS PRODUCT IN BINARY FORMAT (0 -
; 99(DECIMAL) = 0 - 63H)
MOV B,#10 ; DIVIDE PRODUCT BY 10
DIV AB ; A HOLDS # OF TENS, B HOLDS REMAINDER
SWAP A
ORL A,B ; PACK DIGITS
RET
```

Logical Byte Operations — ANL, ORL, XRL

The instructions ANL, ORL, and XRL perform the logical functions AND, OR, and/or Exclusive-OR on the two byte variables indicated, leaving the results in the first. No flags are affected. (A word to the wise — do not vocalize the first two mnemonics in mixed company.)

These operations may use all the same addressing modes as the arithmetics (ADD, etc.) but unlike the arithmetics, they are not restricted to operating on the accumulator. Directly addressed bytes may be used as the destination with either the accumulator or a constant as the source. These instructions are useful for clearing (ANL), setting (ORL), or complementing (XRL) one or more bits in a RAM, output ports, or control registers. The pattern of bits to be affected is indicated by a suitable mask byte. Use immediate addressing when the pattern to be affected is known at assembly time (Figure 14); use the accumulator versions when the pattern is computed at run-time.

I/O ports are often used for parallel data in formats other than simple eight-bit bytes. For example, the low-order five bits of port 1 may output an alphabetic character code (hopefully) without disturbing bits 7-5. This can be a simple two-step process. First, clear the low-order five pins with an ANL instruction; then set those pins corresponding to ones in the accumulator. (This example assumes the three high-order bits of the accumulator are originally zero.)

Example 14—Reconfiguring Port Size with Logical Byte Instructions

```
OUT_PX: ANL P1,#11100000B ; CLEAR BITS P1.4 - P1.0
ORL P1,A ; SET P1 PINS CORRESPONDING TO SET ACC
; BITS.
RET
```

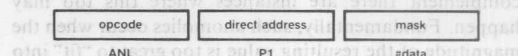


Figure 14. Instruction Pattern for Logical Operation Special Addressing Modes

In this example, low-order bits remaining high may “glitch” low for one machine cycle. If this is undesirable, use a slightly different approach: First, set all pins corresponding to accumulator one bits, then clear the pins corresponding to zeroes in low-order accumulator bits. Not all bits will change from original to final state at the same instant, but no bit makes an intermediate transition.

Example 15 — Reconfiguring I/O Port Size without Glitching

```
ALT_PX: ORL    P1, A
        ORL    A, #11100000B
        ANL    P1, A
        RET
```

Program Control — Jumps, Calls, Returns

Whereas the 8048 only has a single form of the simple jump instruction, the 8051 has three. Each causes the program to unconditionally jump to some other address. They differ in how the machine code represents the destination address.

LJMP (Long Jump) encodes a sixteen-bit address in the second and third instruction bytes (Figure 15.a); the destination may be anywhere in the 64 Kilobyte program memory address space.

The two-byte AJMP (Absolute Jump) instruction encodes its destination using the same format as the 8048: address bits 10 through 8 form a three bit field in the opcode and address bits 7 through 0 form the second byte (Figure 15.b). Address bits 15-12 are unchanged from the (incremented) contents of the P.C., so AJMP can only be used when the destination is known to be within the same 2K memory block. (Otherwise ASM51 will point out the error.)

A different two-byte jump instruction is legal with any nearby destination, regardless of memory block boundaries or “pages.” SJMP (Short Jump) encodes the destination with a program counter-relative address in the second byte (Figure 15.c). The CPU calculates the

destination at run-time by adding the signed eight-bit displacement value to the incremented P.C. Negative offset values will cause jumps up to 128 bytes backwards; positive values up to 127 bytes forwards. (SJMP with 00H in the machine code offset byte will proceed with the following instruction).

In keeping with the 8051 assembly language goal of minimizing the number of instruction mnemonics, there is a “generic” form of the three jump instructions. ASM51 recognizes the mnemonic JMP as a “pseudo-instruction,” translating it into the machine instructions LJMP, AJMP, or SJMP, depending on the destination address.

Like SJMP, all conditional jump instructions use relative addressing. JZ (Jump if Zero) and JNZ (Jump if Not Zero) monitor the state of the accumulator as implied by their names, while JC (Jump on Carry) and JNC (Jump on No Carry) test whether or not the carry flag is set. All four are two-byte instructions, with the same format as Figure 15.c. JB (Jump on Bit), JNB (Jump on No Bit) and JBC (Jump on Bit then Clear Bit) can test any status bit or input pin with a three byte instruction; the second byte specifies which bit to test and the third gives the relative offset value.

There are two subroutine-call instructions, LCALL (Long Call) and ACALL (Absolute Call). Each increments the P.C. to the first byte of the following instruction, then pushes it onto the stack (low byte first). Saving both bytes increments the stack pointer by two. The subroutine’s starting address is encoded in the same ways as LJMP and AJMP. The generic form of the call operation is the mnemonic CALL, which ASM51 will translate into LCALL or ACALL as appropriate.

The return instruction RET pops the high- and low-order bytes of the program counter successively from the stack, decrementing the stack pointer by two. Program execution continues at the address previously pushed; the first byte of the instruction immediately following the call.

When an interrupt request is recognized by the 8051 hardware, two things happen. Program control is automatically “vectored” to one of the interrupt service routine starting addresses by, in effect, forcing the CPU to process an LCALL instead of the next instruction. This automatically stores the return address on the stack. (Unlike the 8048, no status information is automatically saved.)

Secondly, the interrupt logic is disabled from accepting any other interrupts from the same or lower priority. After completing the interrupt service routine, executing an RETI (Return from Interrupt) instruction will return execution to the point where the background program was interrupted — just like RET — while restoring the interrupt logic to its previous state.

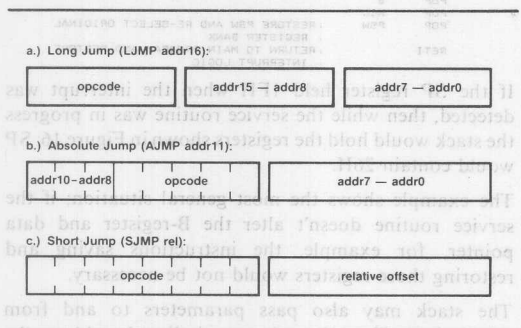


Figure 15. Jump Instruction Machine Code Formats

Operate-and-branch instructions — CJNE, DJNZ

Two groups of instructions combine a byte operation with a conditional jump based on the results.

CJNE (Compare and Jump if Not Equal) compares two byte operands and executes a jump if they disagree. The carry flag is set following the rules for subtraction: if the unsigned integer value of the first operand is less than that of the second it is set; otherwise, it is cleared. However, neither operand is modified.

The CJNE instruction provides, in effect, a one-instruction "case" statement. This instruction may be executed repeatedly, comparing the code variable to a list of "special case" value: the code segment following the instruction (up to the destination label) will be executed only if the operands match. Comparing the accumulator or a register to a series of constants is a convenient way to check for special handling or error conditions; if none of the cases match the program will continue with "normal" processing.

A typical example might be a word processing device which receives ASCII characters through the serial port and drives a thermal hard-copy printer. A standard routine translates "printing" characters to bit patterns, but control characters (, <CR>, <LF>, <BEL>, <ESC> or <SP>) must invoke corresponding special routines. Any other character with an ASCII code less than 20H should be translated into the <NUL> value, 00H, and processed with the printing characters.

Example 16—Case Statements Using CJNE

```
CHAR EQU R7 ; CHARACTER CODE VARIABLE
INTP_1 CJNE CHAR, #7FH, INTP_1 ; (SPECIAL ROUTINE FOR RUBOUT CODE)
RET
INTP_2 CJNE CHAR, #07H, INTP_2 ; (SPECIAL ROUTINE FOR BELL CODE)
RET
INTP_3 CJNE CHAR, #0AH, INTP_3 ; (SPECIAL ROUTINE FOR LFEEED CODE)
RET
INTP_4 CJNE CHAR, #0DH, INTP_4 ; (SPECIAL ROUTINE FOR RETURN CODE)
RET
INTP_5 CJNE CHAR, #1BH, INTP_5 ; (SPECIAL ROUTINE FOR ESCAPE CODE)
RET
INTP_6 CJNE CHAR, #20H, INTP_6 ; (SPECIAL ROUTINE FOR SPACE CODE)
RET
INTP_6 JC PRINTC ; JUMP IF CODE > 20H
MOV CHAR, #0 ; REPLACE CONTROL CHARACTERS WITH
PRINTC ; NULL CODE
PRINTC ; PROCESS STANDARD PRINTING
CHARACTER
RET
```

DJNZ (Decrement and Jump if Not Zero) decrements the register or direct address indicated and jumps if the result is not zero, without affecting any flags. This provides a simple means for executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction. For example, a 99-usec. software delay loop can be added to code forcing an I/O pin low with only two instructions.

Example 17—Inserting a Software Delay with DJNZ

```
CLR R2
MOV R2, #49
DJNZ R2, $
SETB WR
```

The dollar sign in this example is a special character meaning "the address of this instruction." It is useful in eliminating instruction labels on the same or adjacent source lines. CJNE and DJNZ (like all conditional jumps) use program-counter relative addressing for the destination address.

Stack Operations — PUSH, POP

The PUSH instruction increments the stack pointer by one, then transfers the contents of the single byte variable indicated (direct addressing only) into the internal RAM location addressed by the stack pointer. Conversely, POP copies the contents of the internal RAM location addressed by the stack pointer to the byte variable indicated, then decrements the stack pointer by one.

(Stack Addressing follows the same rules, and addresses the same locations as Register-indirect. Future micro-computers based on the MCS-51™ CPU could have up to 256 bytes of RAM for the stack.)

Interrupt service routines must not change any variable or hardware registers modified by the main program, or else the program may not resume correctly. (Such a change might look like a spontaneous random error.) Resources used or altered by the service routine (Accumulator, PSW, etc.) must be saved and restored to their previous value before returning from the service routine. PUSH and POP provide an efficient and convenient way to save register states on the stack.

Example 18—Use of the Stack for Status Saving on Interrupts

```
LOC_TMP EQU $ ; REMEMBER LOCATION COUNTER
ORG 0003H ; STARTING ADDRESS FOR INTERRUPT ROUTINE
LJMP SERVER ; JUMP TO ACTUAL SERVICE ROUTINE LOCATED ELSEWHERE
;
ORG LOC_TMP ; RESTORE LOCATION COUNTER
PUSH PSW ; SAVE ACCUMULATOR (NOTE DIRECT ADDRESSING NOTATION)
PUSH B ; SAVE B REGISTER
PUSH DPL ; SAVE DATA POINTER
PUSH DPH ;
MOV PSW, #00001000B ; SELECT REGISTER BANK 1
;
POP DPH ; RESTORE REGISTERS IN REVERSE ORDER
POP DPL
POP B
POP ACC
POP PSW ; RESTORE PSW AND RE-SELECT ORIGINAL REGISTER BANK
RETI ; RETURN TO MAIN PROGRAM AND RESTORE INTERRUPT LOGIC
```

If the SP register held 1FH when the interrupt was detected, then while the service routine was in progress the stack would hold the registers shown in Figure 16; SP would contain 26H.

The example shows the most general situation; if the service routine doesn't alter the B-register and data pointer, for example, the instructions saving and restoring those registers would not be necessary.

The stack may also pass parameters to and from subroutines. The subroutine can indirectly address the parameters derived from the contents of the stack pointer.

RAM ADDR	
7FH	
26H	DPH ← (SP)
25H	DPL
24H	B
23H	ACC
22H	PSW
21H	PC (HIGH)
20H	PC (LOW)
1FH	
00H	

Figure 16. Stack contents during interrupt

One advantage here is simplicity. Variables need not be allocated for specific parameters; a potentially large number of parameters may be passed, and different calling programs may use different techniques for determining or handling the variables.

For example, the following subroutine reads out a parameter stored on the stack by the calling program, uses the low order bits to access a local look-up table holding bit patterns for driving the coils of a four phase stepper motor, and stores the appropriate bit pattern back in the same position on the stack before returning. The accumulator contents are left unchanged.

Example 19—Passing Variable Parameters to Subroutines Using the Stack

```

NXTPOS: MOV RO, SP      ; ACCESS LOCATION PARAMETER PUSHED INTO
DEC RO      ; ACCUMULATOR
XCH A, R0    ; READ INPUT PARAMETER AND SAVE
ANL A, #03H  ; MASK ALL BUT LOW-ORDER TWO BITS
ADD A, #2    ; ALLOW FOR OFFSET FROM MOVX TO TABLE
MOVC A, @A+PC ; READ LOOK-UP TABLE ENTRY
XCH A, R0    ; PASS BACK TRANSLATED VALUE AND RESTORE
RET          ; RETURN TO BACKGROUND PROGRAM
SPTBL: DB 01101111B ; POSITION 0
DB 01011111B ; POSITION 1
DB 10011111B ; POSITION 2
DB 10101111B ; POSITION 3

```

The background program may reach this subroutine with several different calling sequences, all of which PUSH a value before calling the routine and POP the result after. A motor on Port 1 may be initialized by placing the desired position (zero) on the stack before calling the subroutine and outputting the results directly to a port afterwards.

Example 20—Sending and Receiving Data Parameters Via the Stack

```

CLR A
PUSH A
CALL NXTPOS
POP P1

```

If the position of the motor is determined by the contents of variable POSM1 (a byte in internal RAM) and the position of a second motor on Port 2 is determined by the data input to the low-order nibble of Port 2, a six-instruction sequence could update them both.

Example 21—Loading and Unloading Stack Direct from I/O Ports

```

POSM1 EQU 51
PUSH POSM1
CALL NXTPOS
POP P1
PUSH P2
CALL NXTPOS
POP P2

```

Data Pointer and Table Look-up instructions — MOV, INC, MOVC, JMP

The data pointer can be loaded with a 16-bit value using the instruction MOV DPTR, #data16. The data used is stored in the second and third instruction bytes, high-order byte first. The data pointer is incremented by INC DPTR. A 16-bit increment is performed; an overflow from the low byte will carry into the high-order byte. Neither instruction affects any flags.

The MOVC (Move Constant) instructions (MOVC A, @A+DPTR and MOVC A, @A+PC) read into the accumulator bytes of data from the program memory logical address space. Both use a form of indexed addressing: the former adds the unsigned eight-bit accumulator contents with the sixteen-bit data pointer register, and uses the resulting sum as the address from which the byte is fetched. A sixteen-bit addition is performed; a carry-out from the low-order eight bits may propagate through higher-order bits, but the contents of the DPTR are not altered. The latter form uses the incremented program counter as the "base" value instead of the DPTR (figure 17). Again, neither version affects the flags.

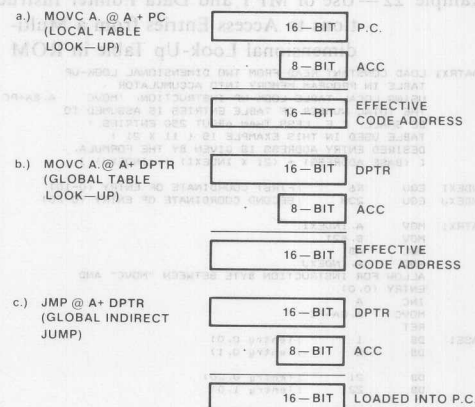


Figure 17. Operation of MOVC instructions

AFN-01502A-25

up tables in ROM. To use the DPTR-relative version, load the Data Pointer with the starting address of a look-up table; load the accumulator with (or compute) the index of the entry desired; and execute `MOVC A,@A+DPTR`. Unlike the similar `MOVP3` instructions in the 8048, the table may be located anywhere in program memory. The data pointer may be loaded with a constant for short tables. Or to allow more complicated data structures, or tables with more than 256 entries, the values for DPH and DPL may be computed or modified with the standard arithmetic instruction set.

The PC-relative version has the advantage of not affecting the data pointer. Again, a look-up sequence takes three steps: load the accumulator with the index; compensate for the offset from the look-up instruction to the start of the table by adding the number of bytes separating them to the accumulator; then execute the `MOVC A,@A+PC` instruction.

Let's look at a non-trivial situation where this instruction would be used. Some applications store large multi-dimensional look-up tables of dot matrix patterns, non-linear calibration parameters, and so on in a linear (one-dimensional) vector in program memory. To retrieve data from the tables, variables representing matrix indices must be converted to the desired entry's memory address. For a matrix of dimensions (MDIMEN x NDIMEN) starting at address BASE and respective indices INDEXI and INDEXJ, the address of element (INDEXI, INDEXJ) is determined by the formula,

$$\text{Entry Address} = \text{BASE} + (\text{NDIMEN} \times \text{INDEXI}) + \text{INDEXJ}$$

The code shown below can access any array with less than 255 entries (i.e., an 11x21 array with 231 elements). The table entries are defined using the Data Byte ("DB") directive, and will be contained in the assembly object code as part of the accessing subroutine itself.

Example 22—Use of MPY and Data Pointer Instructions to Access Entries from a Multi-dimensional Look-Up Table in ROM

```

; MATRX1 LOAD CONSTANT READ FROM TWO DIMENSIONAL LOOK-UP
; TABLE IN PROGRAM MEMORY INTO ACCUMULATOR
; USING LOCAL TABLE LOOK-UP INSTRUCTION. 'MOVC A,@A+PC'
; THE TOTAL NUMBER OF TABLE ENTRIES IS ASSUMED TO
; BE SMALL, I.E. LESS THAN ABOUT 250 ENTRIES.
; TABLE USED IN THIS EXAMPLE IS ( 11 X 21 )
; DESIRED ENTRY ADDRESS IS GIVEN BY THE FORMULA,
; [ (BASE ADDRESS) + (21 X INDEXI) + (INDEXJ) ]
;
INDEXI EQU R6 ; FIRST COORDINATE OF ENTRY (0-10)
INDEXJ EQU 23H ; SECOND COORDINATE OF ENTRY (0-20)
;
MATRX1: MOV A,INDEXI
        MOV B,#21
        MUL AB
        ADD A,A,INDEXJ
        ; ALLOW FOR INSTRUCTION BYTE BETWEEN "MOVC" AND
        ; ENTRY (0,0)
        INC A
        MOVC A,@A+PC
        RET
;
BASE1: DB 1 ; (entry 0,0)
        DB 2 ; (entry 0,1)
;
        DB 21 ; (entry 0,20)
        DB 22 ; (entry 1,0)
;
        DB 42 ; (entry 1,20)
;
        DB 231 ; (entry 10,20)

```

sections of code determined or selected at run time. (The single destination addresses incorporated into conditional and unconditional jumps are, of course, determined at assembly time). Each has advantages for different applications.

The most common is an N-way conditional jump based on some variable, with all of the potential destinations known at assembly time. One of a number of small routines is selected according to the value of an index variable determined while the program is running. The most efficient way to solve this problem is with the `MOVC` and an indirect jump instruction, using a short table of one byte offset values in ROM to indicate the relative starting addresses of the several routines.

`JMP @A+DPTR` is an instruction which performs an indirect jump to an address determined during program execution. The instruction adds the eight-bit unsigned accumulator contents with the contents of the sixteen-bit data pointer, just like `MOVC A,@A+DPTR`. The resulting sum is loaded into the program counter and is used as the address for subsequent instruction fetches. Again, a sixteen-bit addition is performed; a carry out from the low-order eight bits may propagate through the higher-order bits. In this case, neither the accumulator contents nor the data pointer is altered.

The example subroutine below reads a byte of RAM into the accumulator from one of four alternate address spaces, as selected by the contents of the variable MEMSEL. The address of the byte to be read is determined by the contents of R0 (and optionally R1). It might find use in a printing terminal application, where four different model printers all use the same ROM code but use different types and sizes of buffer memory for different speeds and options.

Example 23—N-Way Branch and Computed Jump Instructions via `JMP @ADPTR`

```

MEMSEL EQU R3
JUMP_4: MOV A,MEMSEL
        MOV DPTR,#JMPTBL
        MOVC A,@A+DPTR
        JMP @A+DPTR
;
JMPTBL: DB MEMSP0-JMPTBL
        DB MEMSP1-JMPTBL
        DB MEMSP2-JMPTBL
        DB MEMSP3-JMPTBL
;
MEMSP0: MOV A,R0 ; READ FROM INTERNAL RAM
        RET
MEMSP1: MOVX A,@R0 ; READ FROM 256 BYTES OF EXTERNAL RAM
        RET
MEMSP2: MOV DPL,R0
        MOV DPH,R1
        MOVX A,@DPTR ; READ FROM 64K BYTES OF EXTERNAL RAM
        RET
MEMSP3: MOV A,R1
        ANL A,#07H
        ANL P1,#1111000B
        ORL P1,A
        MOVX A,@R0 ; READ FROM 4K BYTES OF EXTERNAL RAM
        RET

```

Note that this approach is suitable whenever the size of jump table plus the length of the alternate routines is less than 256 bytes. The jump table and routines may be located anywhere in program memory, independent of 256-byte program memory pages.

For applications where up to 128 destinations must be selected, all of which reside in the same 2K page of program memory which may be reached by the two-byte absolute jump instructions, the following technique may be used. In the above mentioned printing terminal example, this sequence could “parse” 128 different codes for ASCII characters arriving via the 8051 serial port.

Example 24 — N-Way Branch with 128 Optional Destinations

```

OPTION EQU R3
JMP128 MOV A, OPTION
RL DPTR, #INSTBL ; MULTIPLY BY 2 FOR 2 BYTE JUMP TABLE
MOV DPTR, #INSTBL ; FIRST ENTRY IN JUMP TABLE
JMP @A+DPTR ; JUMP INTO JUMP TABLE

INSTBL AJMP PROC00 ; 128 CONSECUTIVE
AJMP PROC01 ; JUMP INSTRUCTIONS
AJMP PROC02
...
AJMP PROC7E
AJMP PROC7F

```

The destinations in the jump table (PROC00-PROC7F) are not all necessarily unique routines. A large number of special control codes could each be processed with their own unique routine, with the remaining printing characters all causing a branch to a common routine for entering the character into the output queue.

In those rare situations where even 128 options are insufficient, or where the destination routines may cross a 2K page boundary, the above approach may be modified slightly as shown below.

Example 25 — 256-Way Branch Using Address Look-Up Tables

```

RTMP EQU R7
JMP256 MOV DPTR, #ADRTBL ; FIRST ENTRY IN TABLE OF ADDRESSES
MOV A, OPTION
CLR C
RLC A ; MULTIPLY BY 2 FOR 2 BYTE JUMP TABLE
JNC LOW128
INC DPH
MOV RTMP, A ; SAVE ACC FOR HIGH BYTE READ
MOVC A, @A+DPTR ; READ LOW BYTE FROM JUMP TABLE
XCH A, RTMP
INC A
MOVC A, @A+DPTR ; GET LOW-ORDER BYTE FROM TABLE
PUSH ACC
MOV A, RTMP
MOVC A, @A+DPTR ; GET HIGH-ORDER BYTE FROM TABLE
PUSH ACC
; THE TWO ACC PUSHES HAVE PRODUCED
; A "RETURN ADDRESS" ON THE STACK WHICH CORRESPONDS
; TO THE DESIRED STARTING ADDRESS
; IT MAY BE REACHED BY POPPING THE STACK
; INTO THE PC
RET

ADRTBL DW PROC00 ; UP TO 256 CONSECUTIVE DATA
DW PROC01 ; WORDS INDICATING STARTING ADDRESSES
...
DW PROC7F
DW PROCFF

; DUMMY CODE ADDRESS DEFINITIONS NEEDED BY ABOVE
; TWO EXAMPLES
PROC00 NOP
PROC01 NOP
PROC02 NOP
PROC7E NOP
PROC7F NOP
PROCFF NOP

```

4. BOOLEAN PROCESSING INSTRUCTIONS

The commonly accepted terms for tasks at either end of the computational vs. control application spectrum are, respectively, “number-crunching” and “bit-banging”.

Prior to the introduction of the MCS-51™ family, nice number-crunchers made bad bit-bangers and vice versa. The 8051 is the industry's first single-chip micro-computer designed to crunch **and** bang. (In some circles, the latter technique is also referred to as “bit-twiddling”. Either is correct.)

Direct Bit Addressing

A number of instructions operate on Boolean (one-bit) variables, using a direct bit addressing mode comparable to direct byte addressing. An additional byte appended to the opcode specifies the Boolean variable, I/O pin, or control bit used. The state of any of these bits may be tested for “true” or “false” with the conditional branch instructions JB (Jump on Bit) and JNB (Jump on Not Bit). The JBC (Jump on Bit and Clear) instruction combines a test-for-true with an unconditional clear.

As in direct byte addressing, bit 7 of the address byte switches between two physical address spaces. Values between 0 and 127 (00H-7FH) define bits in internal RAM locations 20H to 2FH (Figure 18a); address bytes between 128 and 255 (80H-0FFH) define bits in the 2 x “special-function” register address space (Figure 18b). If no 2 x “special-function” register corresponds to the direct bit address used the result of the instruction is undefined.

Bits so addressed have many wondrous properties. They may be set, cleared, or complemented with the two byte instructions SETB, CLR, or CPL. Bits may be moved to and from the carry flag with MOV. The logical ANL and ORL functions may be performed between the carry and either the addressed bit or its complement.

Bit Manipulation Instructions — MOV

The “MOV” mnemonic can be used to load an addressable bit into the carry flag (“MOV C, bit”) or to copy the state of the carry to such a bit (“MOV bit, C”). These instructions are often used for implementing serial I/O algorithms via software or to adapt the standard I/O port structure.

It is sometimes desirable to “re-arrange” the order of I/O pins because of considerations in laying out printed circuit boards. When interfacing the 8051 to an immediately adjacent device with “weighted” input pins, such as keyboard column decoder, the corresponding pins are likely to be not aligned (Figure 19).

There is a trade-off in “scrambling” the interconnections with either interwoven circuit board traces or through software. This is extremely cumbersome (if not impossible) to do with byte-oriented computer architectures. The 8051's unique set of Boolean instructions makes it simple to move individual bits between arbitrary locations.

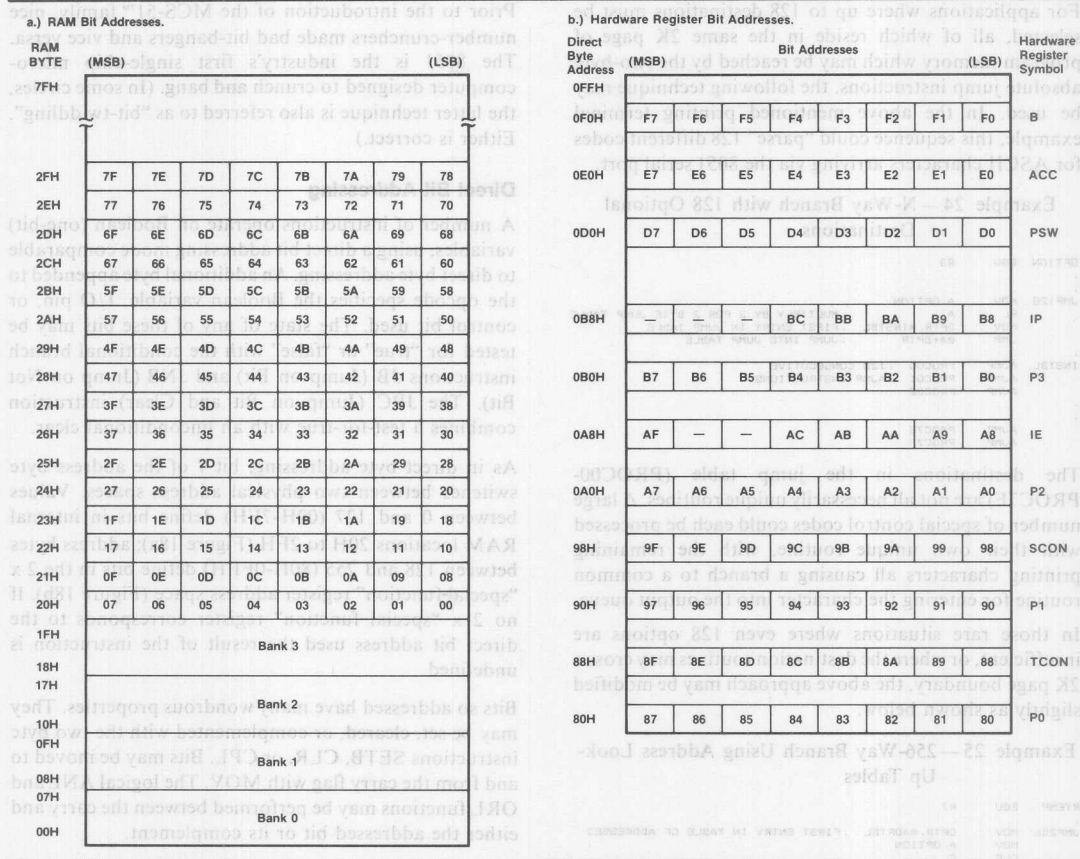


Figure 18. Bit Address Maps

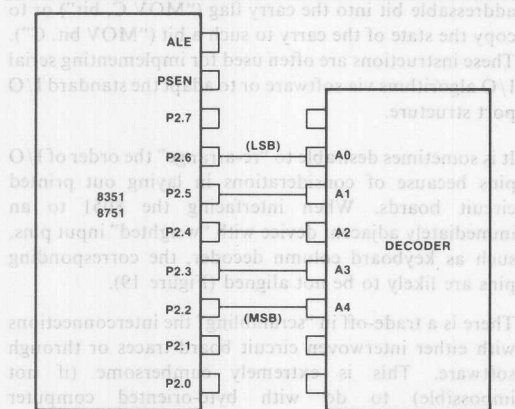


Figure 19. "Mismatch" Between I/O port and Decoder

Example 26—Re-ordering I/O Port Configuration

```

OUT_P2: RRC      A      , MOVE ORIGINAL ACC.0 INTO.CY
MOV      P2.6,C      , STORE CARRY TO PIN P26
RRC      A      , MOVE ORIGINAL ACC.1 INTO.CY
MOV      P2.5,C      , STORE CARRY TO PIN P25
RRC      A      , MOVE ORIGINAL ACC.2 INTO.CY
MOV      P2.4,C      , STORE CARRY TO PIN P24
RRC      A      , MOVE ORIGINAL ACC.3 INTO.CY
MOV      P2.3,C      , STORE CARRY TO PIN P23
RRC      A      , MOVE ORIGINAL ACC.4 INTO.CY
MOV      P2.2,C      , STORE CARRY TO PIN P22
RET

```

Solving Combinatorial Logic Equations — ANL, ORL

Virtually all hardware designers are familiar with the problem of solving complex functions using combinatorial logic. The technologies involved may vary greatly, from multiple contact relay logic, vacuum tubes, TTL, or CMOS to more esoteric approaches like fluidics, but in each case the goal is the same; a Boolean (true/false) function is computed on a number of Boolean variables.

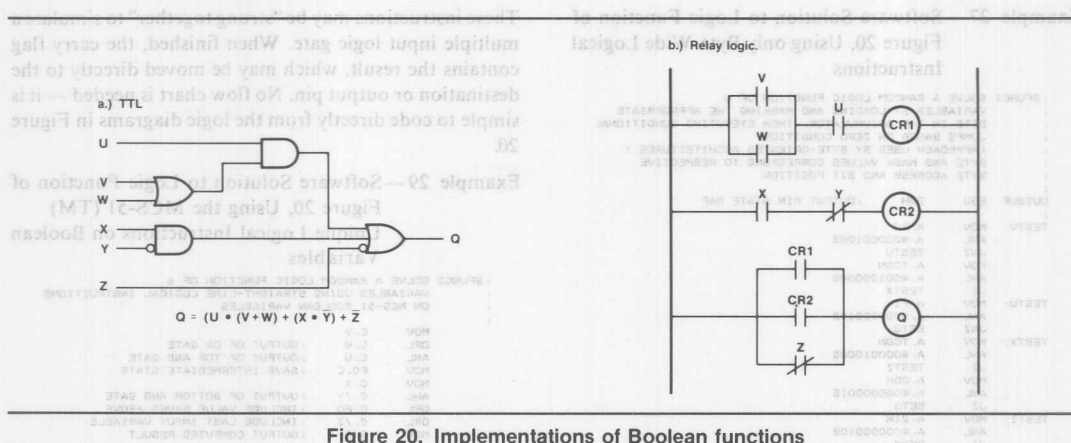


Figure 20. Implementations of Boolean functions

Figure 20 shows the logic diagram for an arbitrary function of six variables named U through Z using standard logic and relay logic symbols. Each is a solution of the equation.

$$Q = (U \cdot (V + W)) + (X \cdot \bar{Y}) + \bar{Z}$$

(While this equation could be reduced using Karnaugh Maps or algebraic techniques, that is not the purpose of this example. Even a minor change to the function equation would require re-reducing from scratch.)

Most digital computers can solve equations of this type with standard word-wide logical instructions and conditional jumps. Still, such software solutions seem somewhat sloppy because of the many paths through the program the computation can take.

Assume U and V are input pins being read by different input ports. W and X are status bits for two peripheral controllers (read as I/O ports), and Y and Z are software flags set or cleared earlier in the program. The end result must be written to an output pin on some third port.

For the sake of comparison we will implement this function with software drawn from three proper subsets of the MCS-51™ instruction set. The first two implementations follow the flow chart shown in Figure 21. Program flow would embark on a route down a test-and-branch tree and leaves either the "True" or "Not True" exit ASAP. These exits then write the output port with the data previously written to the same port with the result bit respectively one or zero.

In the first case, we assume there are no instructions for addressing individual bits other than special flags like the carry. This is typical of many older microprocessors and mainframe computers designed for number-crunching. MCS-51™ mnemonics are used here, though for most other machines the issue would be even further clouded by their use of operation-specific mnemonics like

INPUT, OUTPUT, LOAD, STORE, etc., instead of the universal MOV.

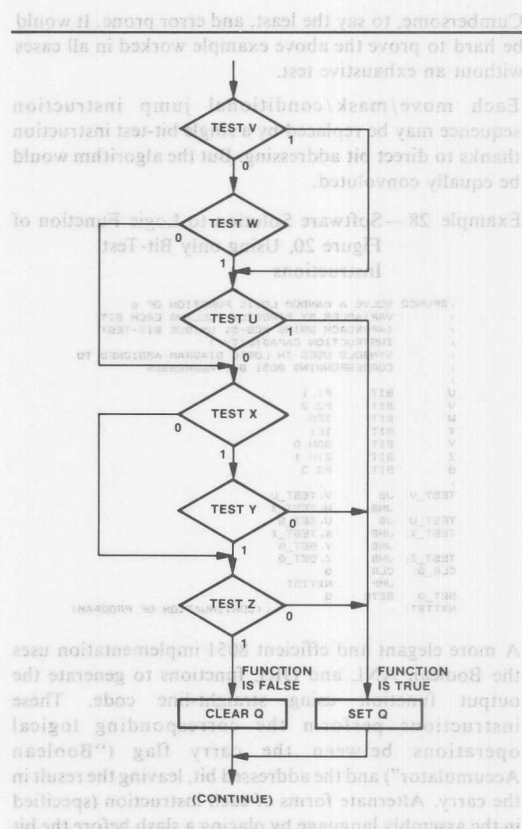


Figure 21. Flow chart for tree-branching logic implementation

Example 27—Software Solution to Logic Function of Figure 20, Using only Byte-Wide Logical Instructions

```

;BFUNC1 SOLVE A RANDOM LOGIC FUNCTION OF 6
; VARIABLES BY LOADING AND MASKING THE APPROPRIATE
; BITS IN THE ACCUMULATOR, THEN EXECUTING CONDITIONAL
; JUMPS BASED ON ZERO CONDITION/
; (APPROACH USED BY BYTE-ORIENTED ARCHITECTURES.)
; BYTE AND MASK VALUES CORRESPOND TO RESPECTIVE
; BYTE ADDRESS AND BIT POSITION.

OUTBUF EQU 22H ; OUTPUT PIN STATE MAP

TESTV: MOV A, P2
ANL A, #00000100B
JNZ TESTU
MOV A, TCON
ANL A, #00100000B
JZ TESTX
TESTU: MOV A, P1
ANL A, #00000010B
JNZ SETG
MOV A, TCON
ANL A, #00000100B
JZ TESTZ
MOV A, 20H
ANL A, #00000001B
JZ SETG
TESTZ: MOV A, 21H
ANL A, #00000001B
JZ SETG
CLRQ: MOV A, OUTBUF
ANL A, #11110111B
JMP OUTG
SETG: MOV A, OUTBUF
ORL A, #00001000B
MOV OUTBUF, A
MOV P3, A

```

Cumbersome, to say the least, and error prone. It would be hard to prove the above example worked in all cases without an exhaustive test.

Each move/mask/conditional jump instruction sequence may be replaced by a single bit-test instruction thanks to direct bit addressing. But the algorithm would be equally convoluted.

Example 28—Software Solution to Logic Function of Figure 20, Using only Bit-Test Instructions

```

;BFUNC2 SOLVE A RANDOM LOGIC FUNCTION OF 6
; VARIABLES BY DIRECTLY POLLING EACH BIT
; (APPROACH USING MCS-51 UNIQUE BIT-TEST
; INSTRUCTION CAPABILITY.)
; SYMBOLS USED IN LOGIC DIAGRAM ASSIGNED TO
; CORRESPONDING 8051 BIT ADDRESSES
;
;
U BIT P1.1
V BIT P2.2
W BIT TFO
X BIT IE1
Y BIT 20H.0
Z BIT 21H.1
Q BIT P3.3

TEST_V: JB V, TEST_U
TEST_U: JB U, SET_Q
TEST_X: JNB X, TEST_Z
TEST_Z: JNB Z, SET_Q
CLR_Q: CLR Q
JMP NXTTST
SET_Q: SETB Q
NXTTST:

```

A more elegant and efficient 8051 implementation uses the Boolean ANL and ORL functions to generate the output function using straight-line code. These instructions perform the corresponding logical operations between the carry flag ("Boolean Accumulator") and the addressed bit, leaving the result in the carry. Alternate forms of each instruction (specified in the assembly language by placing a slash before the bit name) use the complement of the bit's state as the input operand.

These instructions may be "strung together" to simulate a multiple input logic gate. When finished, the carry flag contains the result, which may be moved directly to the destination or output pin. No flow chart is needed—it is simple to code directly from the logic diagrams in Figure 20.

Example 29—Software Solution to Logic Function of Figure 20, Using the MCS-51 (TM) Unique Logical Instructions on Boolean Variables

```

;BFUNC3 SOLVE A RANDOM LOGIC FUNCTION OF 6
; VARIABLES USING STRAIGHT-LINE LOGICAL INSTRUCTIONS
; ON MCS-51 BOOLEAN VARIABLES
;
MOV C, V
ORL C, W ; OUTPUT OF OR GATE
ANL C, U ; OUTPUT OF TOP AND GATE
MOV FO, C ; SAVE INTERMEDIATE STATE
MOV C, X
ANL C, Y ; OUTPUT OF BOTTOM AND GATE
ORL C, FO ; INCLUDE VALUE SAVED ABOVE
ORL C, Z ; INCLUDE LAST INPUT VARIABLE
MOV Q, C ; OUTPUT COMPUTED RESULT

```

Simplicity itself. Fast, flexible, reliable, easy to design, and easy to debug.

The Boolean features are useful and unique enough to warrant a complete Application Note of their own. Additional uses and ideas are presented in Application Note AP-70, **Using the Intel® MCS-51® Boolean Processing Capabilities**, publication number 121519.

5. ON-CHIP PERIPHERAL FUNCTION OPERATION AND INTERFACING

I/O Ports

The I/O port versatility results from the "quasi-bidirectional" output structure depicted in Figure 22. (This is effectively the structure of ports 1, 2, and 3 for normal I/O operations. On port 0 resistor R2 is disabled except during multiplexed bus operations, providing

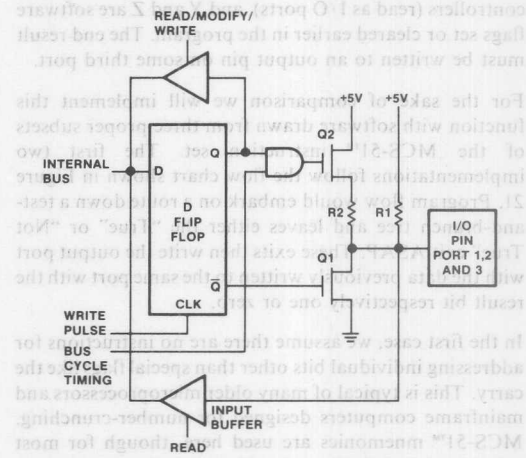


Figure 22. Pseudo-bidirectional I/O port circuitry

AFN-01502A-30

essentially open-collector outputs. For full electrical characteristics see the User's Manual.)

An output latch bit associated with each pin is updated by direct addressing instructions when that port is the destination. The latch state is buffered to the outside world by R1 and Q1, which may drive a standard TTL input. (In TTL terms, Q1 and R1 resemble an open-collector output with a pull-up resistor to Vcc.)

R2 and Q2 represent an "active pull-up" device enabled momentarily when a 0 previously output changes to a 1. This "jerks" the output pin to a 1 level more quickly than the passive pull-up, improving rise-time significantly if the pin is driving a capacitive load. Note that the active pull-up is **only** activated on 0-to-1 transitions at the output latch (unlike the 8048, in which Q2 is activated whenever a 1 is written out).

Operations using an input port or pin as the source operand use the logic level of the pin itself, rather than the output latch contents. This level is affected by both the microcomputer itself and whatever device the pin is connected to externally. The value read is essentially the "OR-tied" function of Q1 and the external device. If the external device is high-impedence, such as a logic gate input or a three state output in the third state, then reading a pin will reflect the logic level previously output. To use a pin for input, the corresponding output latch must be set. The external device may then drive the pin with either a high or low logic signal. Thus the same port may be used as both input and output by writing ones to all pins used as inputs on output operations, and ignoring all pins used as output on an input operation.

In one operand instructions (INC, DEC, DJNZ and the Boolean CPL) the output latch rather than the input pin level is used as the source data. Similarly, two operand instructions using the port as both one source and the destination (ANL, ORL, XRL) use the output latches. This ensures that latch bits corresponding to pins used as inputs will not be cleared in the process of executing these instructions.

The Boolean operation JBC tests the output latch bit, rather than the input pin, in deciding whether or not to jump. Like the byte-wise logical operations, Boolean operations which modify individual pins of a port leave the other bits of the output latch unchanged.

A good example of how these modes may play together may be taken from the host-processor interface expected by an 8243 I/O expander. Even though the 8051 does not include 8048-type instructions for interfacing with an 8243, the parts can be interconnected (Figure 23) and the protocol may be emulated with simple software.

Example 30—Mixing Parallel Output, Input, and Control Strobes on Port 2

```

;INB243 INPUT DATA FROM AN 8243 1-/0 EXPANDER
;      CONNECTED TO P23-P20
;      P25 & P24 MIMIC CS-/ & PROG
;      P27-P26 USED AS INPUTS
;      PORT TO BE READ IN ACC

INB243  OPL  A, #11010000B
        MOV  P2, A ; OUTPUT INSTRUCTION CODE
        CLR  P2, 4 ; FALLING EDGE OF PROG
        ORL  P2, #00001111B ; SET FOR INPUT
        MOV  A, P2 ; READ INPUT DATA
        SETB P2, 4 ; RETURN PROG HIGH
        SETB P2, 5 ; DE-SELECT CHIP

```

Serial Port and Timer applications

Configuring the 8051's Serial Port for a given data rate and protocol requires essentially three short sections of software. On power-up or hardware reset the serial port and timer control words must be initialized to the appropriate values. Additional software is also needed in the transmit routine to load the serial port data register and in the receive routine to unload the data as it arrives. This is best illustrated through an arbitrary example. Assume the 8051 will communicate with a CRT operating at 2400 baud (bits per second). Each character is transmitted as seven data bits, odd parity, and one stop bit. This results in a character rate of 2400/10=240 characters per second.

For the sake of clarity, the transmit and receive subroutines are driven by simple-minded software status polling code rather than interrupts. (It might help to refer back to Figures 7-9 showing the control word formats.) The serial port must be initialized to 8-bit UART mode (M0, M1=01), enabled to receive all messages (M2=0, REN=1). The flag indicating that the transmit register is free for more data will be artificially set in order to let the output software know the output register is available. This can all be set up with one instruction.

Example 31—Serial Port Mode and Control Bits

```

;SPINIT INITIALIZE SERIAL PORT
;      FOR 8-BIT UART MODE
;      & SET TRANSMIT READY FLAG.

SPINIT: MOV  RST, SCON, #01010010B ; M1, M0
        BORR  RST, SCON, #00000001B ; TR

```

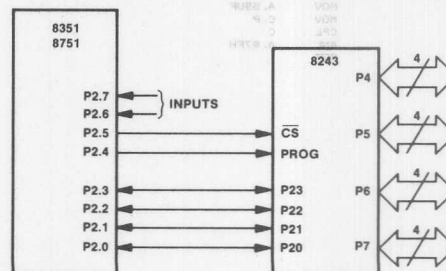


Figure 23. Connecting an 8051 with an 8243 I/O Expander

Timer 1 will be used in auto-reload mode as a data rate generator. To achieve a data rate of 2400 baud, the timer must divide the 1 MHz internal clock by 32 x (desired data rate):

$$\frac{1 \times 10^6}{(32) (2400)}$$

which equals 13.02 rounded down to 13 instruction cycles. The timer must reload the value -13, or 0F3H. (ASM51 will accept both the signed decimal or hexadecimal representations.)

Example 32—Initializing Timer Mode and Control Bits

```

;T1INIT: INITIALIZE TIMER 1 FOR
;AUTO-RELOAD AT 32x2400 HZ
; (T0 USED AS GATED, 16-BIT, COUNTER)
T1INIT: MOV     TCON, #11010010B
        MOV     TH1, #-13
        SETB    TR1

```

A simple subroutine to transmit the character passed to it in the accumulator must first compute the parity bit, insert it into the data byte, wait until the transmitter is available, output the character, and return. This is nearly as easy said as done.

Example 33—Code for UART Output, Adding Parity, Transmitter Loading

```

;SP_OUT: ADD ODD PARITY TO ACC AND
;TRANSMIT WHEN SERIAL PORT READY
SP_OUT: MOV     C, P
        MOV     ACC, 7, C
        JNB     TI, $
        CLR     TI
        MOV     SBUF, A
        RET

```

A simple minded routine to wait until a character is received, set the carry flag if there is an odd-parity error, and return the masked seven-bit code in the accumulator is equally short.

Example 34—Code for UART Reception and Parity Verification

```

;SP_IN: INPUT NEXT CHARACTER FROM SERIAL PORT
;SET CARRY IFF ODD-PARITY ERROR

```

```

SP_IN:  JNB     RI, $
        CLR     RI
        MOV     A, SBUF
        MOV     C, P
        CPL     C
        ANL     A, #7FH
        RET

```

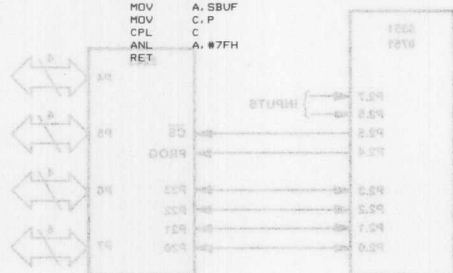


Figure 23. Connecting an 8051 with an 8243 I/O Expander

6. SUMMARY

This Application Note has described the architecture, instruction set, and on-chip peripheral features of the first three members of the MCS-51™ microcomputer family. The examples used throughout were admittedly (and necessarily) very simple. Additional examples and techniques may be found in the MCS-51™ User's Manual and other application notes written for the MCS-48™ and MCS-51™ families.

Since its introduction in 1977, the MCS-48™ family has become the industry standard single-chip microcomputer. The MCS-51™ architecture expands the addressing capabilities and instruction set of its predecessor while ensuring flexibility for the future, and maintaining basic software compatibility with the past.

Designers already familiar with the 8048 or 8049 will be able to take with them the education and experience gained from past designs as ever-increasing system performance demands force them to move on to state-of-the-art products. Newcomers will find the power and regularity of the 8051 instruction set an advantage in streamlining both the learning and design processes.

Microcomputer system designers will appreciate the 8051 as basically a single-chip solution to many problems which previously required board-level computers. Designers of real-time control systems will find the high execution speed, on-chip peripherals, and interrupt capabilities vital in meeting the timing constraints of products previously requiring discrete logic designs. And designers of industrial controllers will be able to convert ladder diagrams directly from tested-and-true TTL or relay-logic designs to microcomputer software, thanks to the unique Boolean processing capabilities.

It has not been the intent of this note to gloss over the difficulty of designing microcomputer-based systems. To be sure, the hardware and software design aspects of any new computer system are nontrivial tasks. However, the system speed and level of integration of the MCS-51™ microcomputers, the power and flexibility of the instruction set, and the sophisticated assembler and other support products combine to give both the hardware and software designer as much of a head start on the problem as possible.

A good example of how these modes may play together may be taken from the host-processor interface expected by an 8243 I/O expander. Even though the 8051 does not include 8048-type instructions for interacting with an 8243, the parts can be interconnected (Figure 23) and the protocol may be simulated with simple software.

AFN-01502A-32

Using the Contents

Intel MCS-51[®] Boolean Processing Capabilities

The MCS-51 is one of the industry's most popular microcontrollers. It is a single-chip microcomputer that integrates a central processing unit (CPU), memory, and peripheral control logic on a single chip. The MCS-51 is designed for use in a wide variety of applications, from simple control systems to complex data processing systems. The MCS-51 is a 8-bit microcontroller, meaning it can process data in 8-bit increments. It has a built-in 2K bytes of EPROM (Erasable Programmable Read-Only Memory) and 2K bytes of RAM (Random Access Memory). The MCS-51 is also capable of interfacing with a variety of external devices, including keyboards, displays, and sensors.

The MCS-51 incorporates a number of special features which support the direct manipulation and testing of individual bits and allow the use of single-bit variables in performing logical operations. Taken together, these features are referred to as the MCS-51's Boolean Processor. While the bit-processing capabilities alone would be adequate to solve many control applications, their true power comes when they are used in conjunction with the microcontroller's byte-processing and numerical capabilities.

Many concepts embodied by the Boolean Processor will certainly be new even to experienced microcomputer system designers. The purpose of this Application Note is to explain these concepts and show how they are used. It is assumed the reader has read Application Note AP-69, "An Introduction to the Intel[®] MCS-51 Single-Chip Microcomputer Family," publication number 131218, or has been exposed to Intel's single-chip microcomputer product lines.

For detailed information on these parts refer to the Intel MCS-51 Family User's Manual, publication number 131217. The instruction set, assembly language, and use of the 8051 assembler (ASM51) are further described in the MCS-51 Macro Assembler User's Guide, publication number 980097.

2. BOOLEAN PROCESSOR OPERATION

The Boolean processing capabilities of the 8051 are based on concepts which have been around for some time. Digital computer systems of widely varying design all have four functional elements in common (Figure 2):

Table 1. Features of Intel's Single-Chip Microcomputers.

Reg. Banks	Interrupt Sources	Input/Output Pins	Inst. Cycle Time	Data Memory (Bytes)	Program Memory (Int/Max)	External Program Memory	ROM Program Memory	EPROM Program Memory
1	0	31	10 μ sec	64	1K-1K	—	8051	—
1	2	38	10 μ sec	64	2K-2K	—	8052	—
2	2	37	2.5 μ sec	64	1K-4K	8032	8048	8748
2	2	37	1.5 μ sec	128	2K-4K	8039	8049	—
4	2	32	1.0 μ sec	128	4K-64K	8051	8051	8751

01-0834-00

1. INTRODUCTION

The Intel microcontroller family now has three new members—the Intel® 8031, 8051, and 8751 single-chip microcomputers. These devices, shown in Figure 1, will allow whole new classes of products to benefit from recent advances in Integrated Electronics. Thanks to Intel's new HMOS® technology, they provide larger program and data memory spaces, more flexible I/O and peripheral capabilities, greater speed, and lower system cost than any previous-generation single-chip microcomputer.

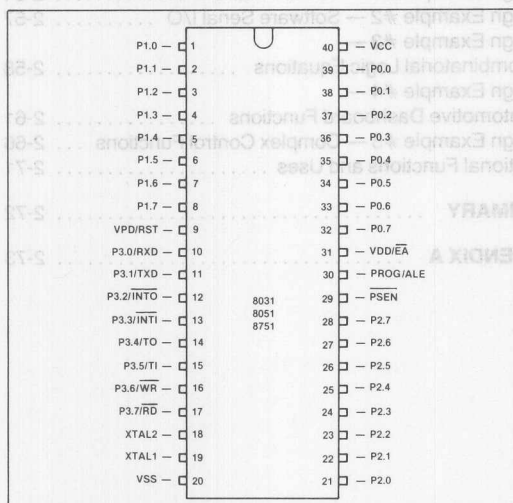


Figure 1. 8051 Family Pinout Diagram.

Table 1 summarizes the quantitative differences between the members of the MCS-48™ and 8051 families. The 8751 contains 4K bytes of EPROM program memory fabricated on-chip, while the 8051 replaces the EPROM with 4K bytes of lower-cost mask-programmed ROM. The 8031 has no program memory on-chip; instead, it accesses up to 64K bytes of program memory from external memory. Otherwise, the three new family members are identical. Throughout this Note, the term "8051" will represent all members of the 8051 Family, unless specifically stated otherwise.

Table 1. Features of Intel's Single-chip Microcomputers.

EPROM Program Memory	ROM Program Memory	External Program Memory	Program Memory (Int/Max)	Data Memory (Bytes)	Instr. Cycle Time	Input/Output Pins	Interrupt Sources	Reg. Banks
—	8021	—	1K / 1K	64	10 μ Sec	21	0	1
—	8022	—	2K / 2K	64	10 μ Sec	28	2	1
8748	8048	8035	1K / 4K	64	2.5 μ Sec	27	2	2
—	8049	8039	2K / 4K	128	1.36 μ Sec	27	2	2
8751	8051	8031	4K / 64K	128	1.0 μ Sec	32	5	4

The CPU in each microcomputer is one of the industry's fastest and most efficient for numerical calculations on byte operands. But controllers often deal with bits, not bytes: in the real world, switch contacts can only be open or closed, indicators should be either lit or dark, motors are either turned on or off, and so forth. For such control situations the most significant aspect of the MCS-51™ architecture is its complete hardware support for one-bit, or *Boolean* variables (named in honor of Mathematician George Boole) as a separate data type.

The 8051 incorporates a number of special features which support the direct manipulation and testing of individual bits and allow the use of single-bit variables in performing logical operations. Taken together, these features are referred to as the MCS-51™ *Boolean Processor*. While the bit-processing capabilities alone would be adequate to solve many control applications, their true power comes when they are used in conjunction with the microcomputer's byte-processing and numerical capabilities.

Many concepts embodied by the Boolean Processor will certainly be new even to experienced microcomputer system designers. The purpose of this Application Note is to explain these concepts and show how they are used. It is assumed the reader has read Application Note AP-69, **An Introduction to the Intel® MCS-51™ Single-Chip Microcomputer Family**, publication number 121518, or has been exposed to Intel's single-chip microcomputer product lines.

For detailed information on these parts refer to the **Intel MCS-51™ Family User's Manual**, publication number 121517. The instruction set, assembly language, and use of the 8051 assembler (ASM51) are further described in the **MCS-51™ Macro Assembler User's Guide**, publication number 9800937.

2. BOOLEAN PROCESSOR OPERATION

The Boolean Processing capabilities of the 8051 are based on concepts which have been around for some time. Digital computer systems of widely varying designs all have four functional elements in common (Figure 2):

- a central processor (CPU) with the control, timing, and logic circuits needed to execute stored instructions;
- a memory to store the sequence of instructions making up a program or algorithm;
- data memory to store variables used by the program; and
- some means of communicating with the outside world.

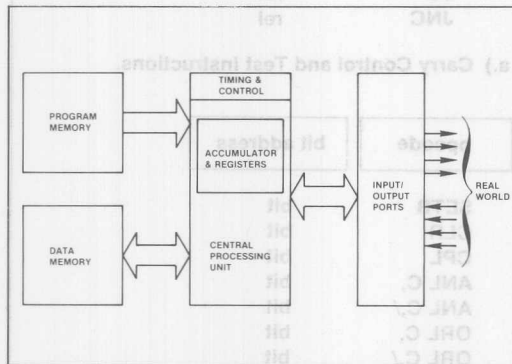


Figure 2. Block Diagram for Abstract Digital Computer.

The CPU usually includes one or more accumulators or special registers for computing or storing values during program execution. The instruction set of such a processor generally includes, at a minimum, operation classes to perform arithmetic or logical functions on program variables, move variables from one place to another, cause program execution to jump or conditionally branch based on register or variable states, and instructions to call and return from subroutines. The program and data memory functions sometimes share a single memory space, but this is not always the case. When the address spaces are separated, program and data memory need not even have the same basic word width.

A digital computer's flexibility comes in part from combining simple fast operations to produce more complex (albeit slower) ones, which in turn link together eventually solving the problem at hand. A four-bit CPU executing multiple precision subroutines can, for example, perform 64-bit addition and subtraction. The subroutines could in turn be building blocks for floating-point multiplication and division routines. Eventually, the four-bit CPU can simulate a far more complex "virtual" machine.

In fact, *any* digital computer with the above four functional elements can (given time) complete *any* algorithm (though the proverbial room full of chimpanzees at word

processors might first re-create Shakespeare's classics and this Application Note)! This fact offers little consolation to product designers who want programs to run as quickly as possible. By definition, a real-time control algorithm *must* proceed quickly enough to meet the preordained speed constraints of other equipment.

One of the factors determining how long it will take a microcomputer to complete a given chore is the number of instructions it must execute. What makes a given computer architecture particularly well- or poorly-suited for a class of problems is how well its instruction set matches the tasks to be performed. The better the "primitive" operations correspond to the steps taken by the control algorithm, the lower the number of instructions needed, and the quicker the program will run. All else being equal, a CPU supporting 64-bit arithmetic directly could clearly perform floating-point math faster than a machine bogged-down by multiple-precision subroutines. In the same way, direct support for bit manipulation naturally leads to more efficient programs handling the binary input and output conditions inherent in digital control problems.

Processing Elements

The introduction stated that the 8051's bit-handling capabilities alone would be sufficient to solve some control applications. Let's see how the four basic elements of a digital computer - a CPU with associated registers, program memory, addressable data RAM, and I/O capability - relate to Boolean variables.

CPU. The 8051 CPU incorporates special logic devoted to executing several bit-wide operations. All told, there are 17 such instructions, all listed in Table 2. Not shown are 94 other (mostly byte-oriented) 8051 instructions.

Program Memory. Bit-processing instructions are fetched from the same program memory as other arithmetic and logical operations. In addition to the instructions of Table 2, several sophisticated program control features like multiple addressing modes, subroutine nesting, and a two-level interrupt structure are useful in structuring Boolean Processor-based programs.

Boolean instructions are one, two, or three bytes long, depending on what function they perform. Those involving only the carry flag have either a single-byte opcode or an opcode followed by a conditional-branch destination byte (Figure 3.a). The more general instructions add a "direct address" byte after the opcode to specify the bit affected, yielding two or three byte encodings (Figure 3.b). Though this format allows potentially 256 directly addressable bit locations, not all of them are implemented in the 8051 family.

Mnemonic	Description	Byte	Cyc
SETB C	Set Carry flag	1	1
SETB bit	Set direct Bit	2	1
CLR C	Clear Carry flag	1	1
CLR bit	Clear direct bit	2	1
CPL C	Complement Carry flag	1	1
CPL bit	Complement direct bit	2	1
MOV C,bit	Move direct bit to Carry flag	2	1
MOV bit,C	Move Carry flag to direct bit	2	2
ANL C,bit	AND direct bit to Carry flag	2	2
ANL C, bit	AND complement of direct bit to Carry flag	2	2
ORL C,bit	OR direct bit to Carry flag	2	2
ORL C, bit	OR complement of direct bit to Carry flag	2	2
JC rel	Jump if Carry is flag is set	2	2
JNC rel	Jump if No Carry flag	2	2
JB bit,rel	Jump if direct Bit set	3	2
JNB bit,rel	Jump if direct Bit Not set	3	2
JBC bit,rel	Jump if direct Bit is set & Clear bit	3	2

Address mode abbreviations:

C — Carry flag.

bit — 128 software flags, any I/O pin, control or status bit

rel — All conditional jumps include an 8-bit offset byte. Range is +127/-128 bytes relative to first byte of the following instruction.

All mnemonics copyrighted© Intel Corporation 1980

Data Memory. The instructions in Figure 3.b can operate directly upon 144 general purpose bits forming the Boolean processor “RAM.” These bits can be used as software flags or to store program variables. Two operand instructions use the CPU’s carry flag (“C”) as a special one-bit register; in a sense, the carry is a “Boolean accumulator” for logical operations and data transfers.

Input/Output. All 32 I/O pins can be addressed as individual inputs, outputs, or both, in any combination. Any pin can be a control strobe output, status (Test) input, or serial I/O link implemented via software. An additional 33 individually addressable bits reconfigure, control, and monitor the status of the CPU and all on-chip peripheral functions (timer/counters, serial port modes, interrupt logic, and so forth).

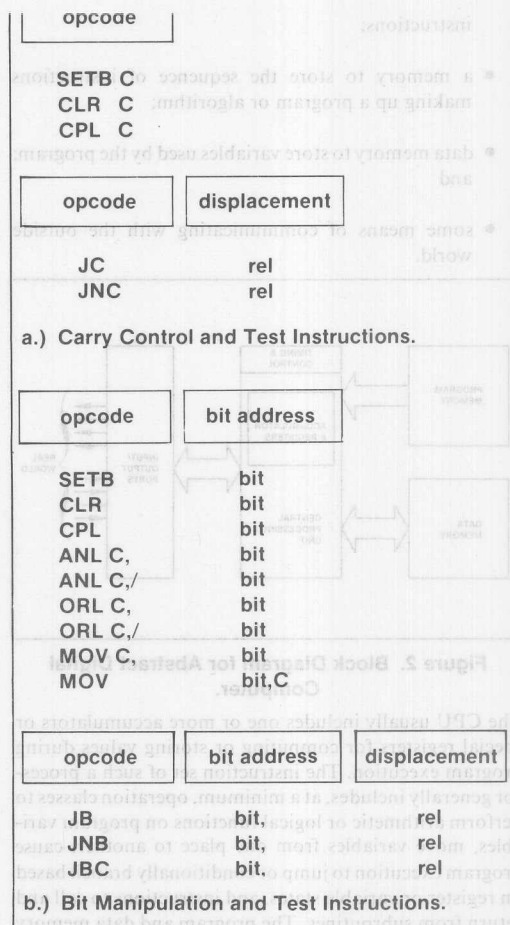


Figure 3. Bit Addressing Instruction Formats.

Direct Bit Addressing

The most significant bit of the direct address byte selects one of two groups of bits. Values between 0 and 127 (00H and 7FH) define bits in a block of 32 bytes of on-chip RAM, between RAM addresses 20H and 2FH (Figure 4.a). They are numbered consecutively from the lowest-order byte’s lowest-order bit through the highest-order byte’s highest-order bit.

Bit addresses between 128 and 255(80H and 0FFH) correspond to bits in a number of special registers, mostly used for I/O or peripheral control. These positions are numbered with a different scheme than RAM: the five high-order address bits match those of the register’s own address, while the three low-order bits identify the bit position within that register (Figure 4.b).

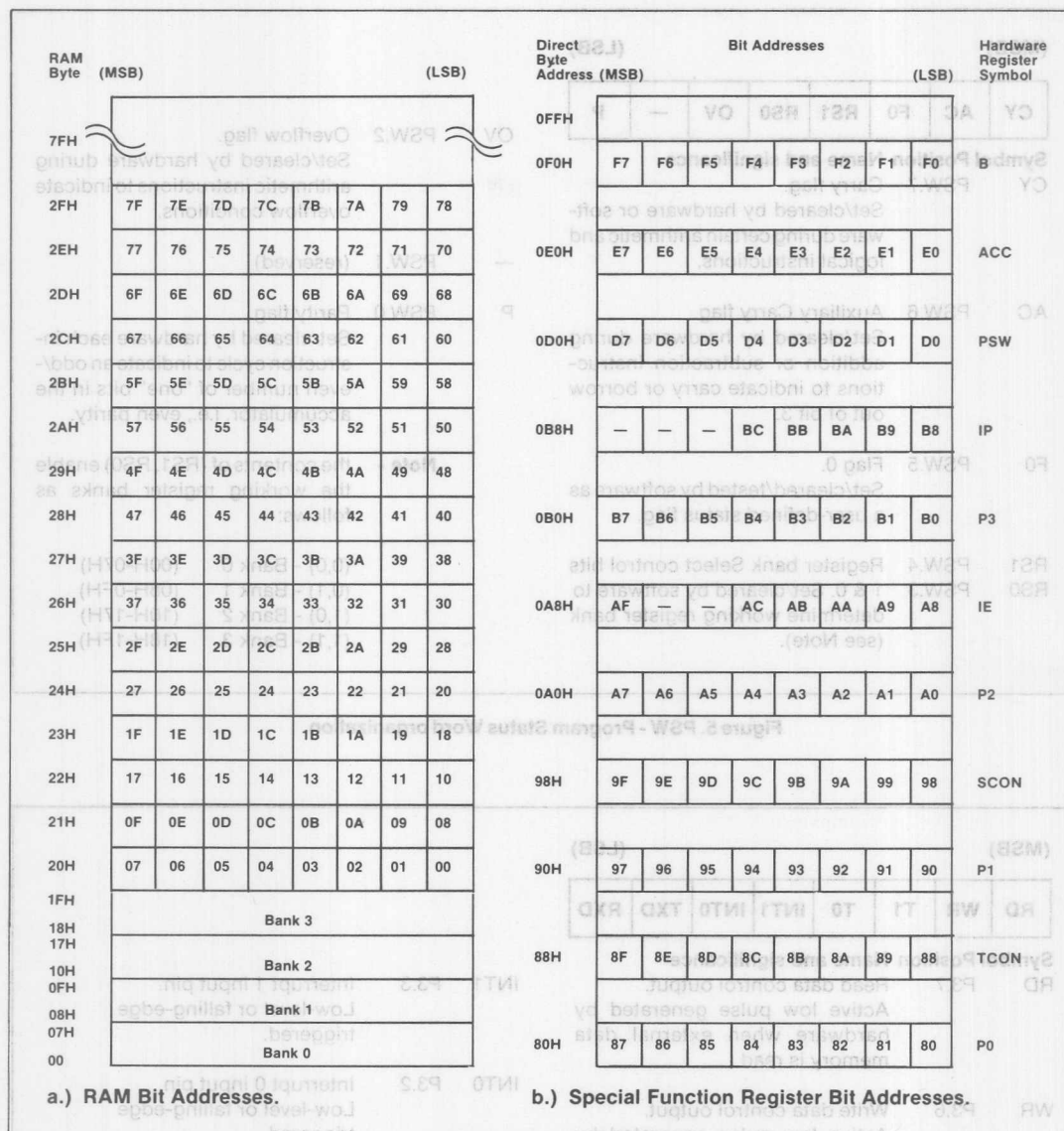


Figure 4. Bit Address Maps.

Notice the column labeled "Symbol" in Figure 5. Bits with special meanings in the PSW and other registers have corresponding symbolic names. General-purpose (as opposed to carry-specific) instructions may access the carry like any other bit by using the mnemonic CY in place of C. P0, P1, P2, and P3 are the 8051's four I/O ports; secondary functions assigned to each of the eight pins of P3 are shown in Figure 6.

Figure 7 shows the last four bit addressable registers. TCON (Timer Control) and SCON (Serial port Control) control and monitor the corresponding peripherals, while IE (Interrupt Enable) and IP (Interrupt Priority) enable and prioritize the five hardware interrupt sources. Like the reserved hardware register addresses, the five bits not implemented in IE and IP should not be accessed; they can not be used as software flags.

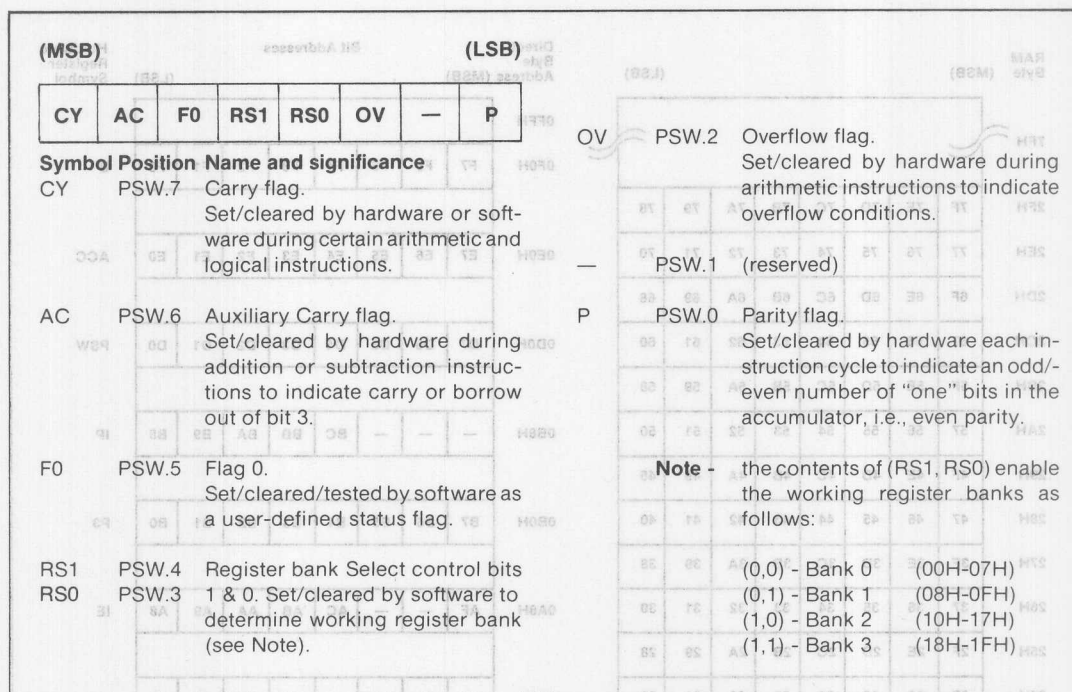


Figure 5. PSW - Program Status Word organization.

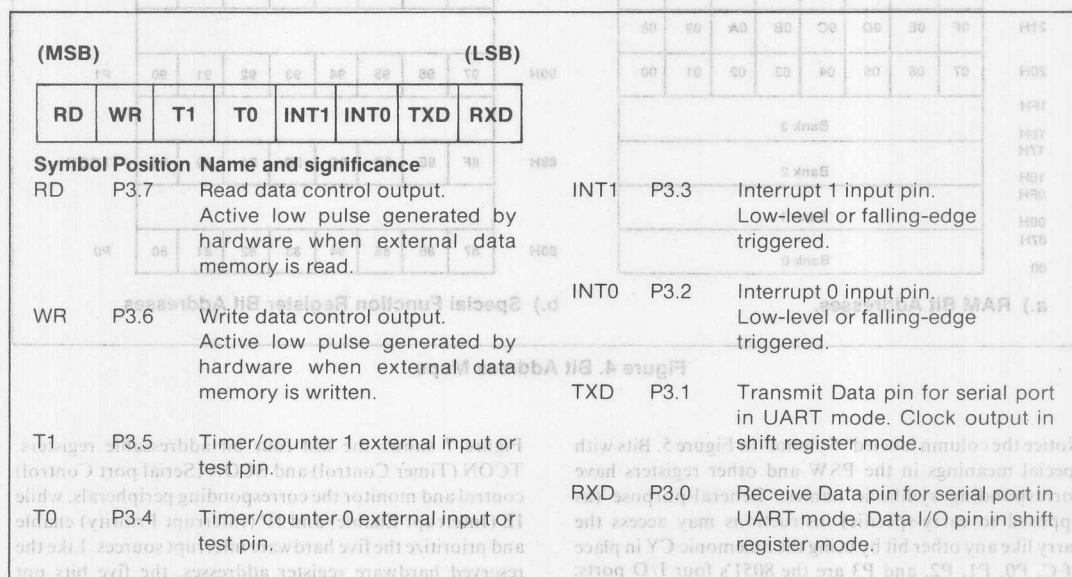


Figure 6. P3 - Alternate I/O Functions of Port 3.

(MSB)				(LSB)			
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Symbol Position Name and significance

TF1	TCON.7	Timer 1 overflow Flag	Set by hardware on timer/counter overflow. Cleared when interrupt processed.				
TR1	TCON.6	Timer 1 Run control bit.	Set/cleared by software to turn timer/counter on/off.				
TF0	TCON.5	Timer 0 overflow Flag.	Set by hardware on timer/counter overflow. Cleared when interrupt processed.				
TR0	TCON.4	Timer 0 Run control bit.	Set/cleared by software to turn timer/counter on/off.				

a.) TCON - Timer/Counter Control/status register.

(MSB)				(LSB)			
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Symbol Position Name and significance

SM0	SCON.7	Serial port Mode control bit 0.	Set/cleared by software (see note).				
SM1	SCON.6	Serial port Mode control bit 1.	Set/cleared by software (see note).				
SM2	SCON.5	Serial port Mode control bit 2.	Set by software to disable reception of frames for which bit 8 is zero.				
REN	SCON.4	Receiver Enable control bit.	Set/cleared by software to enable/disable serial data reception.				
TB8	SCON.3	Transmit Bit 8.	Set/cleared by hardware to determine state of ninth data bit transmitted in 9-bit UART mode.				

b.) SCON - Serial Port Control/status register.

(LSB)				(MSB)			
IE1	IT1	IE0	IT0	EA	EA	EA	EA

IE1	TCON.3	Interrupt 1 Edge flag.	Set by hardware when external interrupt edge detected. Cleared when interrupt processed.				
IT1	TCON.2	Interrupt 1 Type control bit.	Set/cleared by software to specify falling edge/low level triggered external interrupts.				
IE0	TCON.1	Interrupt 0 Edge flag.	Set by hardware when external interrupt edge detected. Cleared when interrupt processed.				
IT0	TCON.0	Interrupt 0 Type control bit.	Set/cleared by software to specify falling edge/low level triggered external interrupts.				

(LSB)				(MSB)			
PX0	PX1	PX2	PX3	PX4	PX5	PX6	PX7

RB8	SCON.2	Receive Bit 8.	Set/cleared by hardware to indicate state of ninth data bit received.				
TI	SCON.1	Transmit Interrupt flag.	Set by hardware when byte transmitted. Cleared by software after servicing.				
RI	SCON.0	Receive Interrupt flag.	Set by hardware when byte received. Cleared by software after servicing.				

Note - the state of (SM0, SM1) selects:
 (0,0) - Shift register I/O expansion.
 (0,1) - 8 bit UART, variable data rate.
 (1,0) - 9 bit UART, fixed data rate.
 (1,1) - 9 bit UART, variable data rate.

Figure 7. Peripheral Configuration Registers.

(MSB)

(LSB)

EA	—	—	ES	ET1	EX1	ET1	EX0
----	---	---	----	-----	-----	-----	-----

Symbol Position Name and significance

EA IE.7 Enable All control bit.
Cleared by software to disable all interrupts, independent of the state of IE.4 - IE.0.

— IE.6 (reserved)
— IE.5

ES IE.4 Enable Serial port control bit.
Set/cleared by software to enable/ disable interrupts from TI or RI flags.

ET1 IE.3 Enable Timer 1 control bit.
Set/cleared by software to enable/ disable interrupts from timer/counter 1.

c.) IE - Interrupt Enable Register.

(MSB)

(LSB)

—	—	—	PS	PT1	PX1	PT0	PX0
---	---	---	----	-----	-----	-----	-----

Symbol Position Name and significance

— IP.7 (reserved)
— IP.6 (reserved)
— IP.5 (reserved)

PS IP.4 Serial port Priority control bit.
Set/cleared by software to specify high/low priority interrupts for Serial port.

PT1 IP.3 Timer 1 Priority control bit.
Set/cleared by software to specify high/low priority interrupts for timer/counter 1.

d.) IP - Interrupt Priority Control Register.**Figure 7. (continued)**

Addressable Register Set. There are 20 special function registers in the 8051, but the advantages of bit addressing only relate to the 11 described below. Five potentially bit-addressable register addresses (0C0H, 0C8H, 0D8H, 0E8H, & 0F8H) are being reserved for possible future expansion in microcomputers based on the MCS-51™ architecture. Reading or writing non-existent registers in the 8051 series is pointless, and may cause unpredictable results. Byte-wide logical operations can be used to manipulate bits in all *non*-bit addressable registers and RAM.

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
-----	-----	-----	-----	-----	-----	-----	-----

EX1 IE.2 Enable External interrupt 1 control bit. Set/cleared by software to enable/disable interrupts from INT1.

ET0 IE.1 Enable Timer 0 control bit.
Set/cleared by software to enable/ disable interrupts from timer/counter 0.

EX0 IE.0 Enable External interrupt 0 control bit. Set/cleared by software to enable/disable interrupts from INT0.

PX1	IP.2	External interrupt 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INT1.
-----	------	--

PT0 IP.1 Timer 0 Priority control bit.
Set/cleared by software to specify high/low priority interrupts for timer/counter 0.

PX0 IP.0 External interrupt 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INT0.

The accumulator and B registers (A and B) are normally involved in byte-wide arithmetic, but their individual bits can also be used as 16 general software flags. Added with the 128 flags in RAM, this gives 144 general purpose variables for bit-intensive programs. The program status word (PSW) in Figure 5 is a collection of flags and machine status bits including the carry flag itself. Byte operations acting on the PSW can therefore affect the carry.

Instruction Set

Having looked at the bit variables available to the Boolean Processor, we will now look at the four classes of instructions that manipulate these bits. It may be helpful to refer back to Table 2 while reading this section.

State Control. Addressable bits or flags may be set, cleared, or logically complemented in one instruction cycle with the two-byte instructions SETB, CLR, and CPL. (The "B" affixed to SETB distinguishes it from the assembler "SET" directive used for symbol definition.) SETB and CLR are analogous to loading a bit with a constant: 1 or 0. Single byte versions perform the same three operations on the carry.

The MCS-51™ assembly language specifies a bit address in any of three ways:

- by a number or expression corresponding to the direct bit address (0-255);
- by the name or address of the register containing the bit, the *dot operator* symbol (a period: "."), and the bit's position in the register (7-0);
- in the case of control and status registers, by the predefined assembler symbols listed in the first columns of Figures 5-7.

Bits may also be given user-defined names with the assembler "BIT" directive and any of the above techniques. For example, bit 5 of the PSW may be cleared by any of the four instructions.

USR_FLG BIT	PSW.5	: User Symbol Definition
CLR	0D5H	: Absolute Addressing
CLR	PSW.5	: Use of Dot Operator
CLR	F0	: Pre-Defined Assembler Symbol
CLR	USR_FLG	: User-Defined Symbol

Data Transfers. The two-byte MOV instructions can transport any addressable bit to the carry in one cycle, or copy the carry to the bit in two cycles. A bit can be moved between two arbitrary locations via the carry by combining the two instructions. (If necessary, push and pop the PSW to preserve the previous contents of the carry.) These instructions can replace the multi-instruction sequence of Figure 8, a program structure appearing in controller applications whenever flags or outputs are conditionally switched on or off.

Logical Operations. Four instructions perform the logical-AND and logical-OR operations between the carry and another bit, and leave the results in the carry. The instruction mnemonics are ANL and ORL; the absence or presence of a

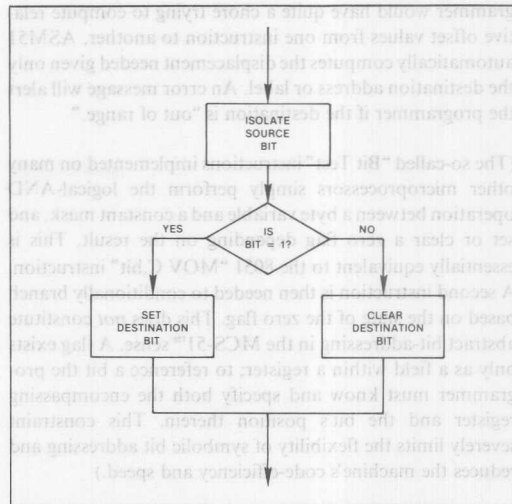


Figure 8. Bit Transfer Instruction Operation.

slash mark ("/) before the source operand indicates whether to use the positive-logic value or the logical complement of the addressed bit. (The source operand itself is never affected.)

Bit-test Instructions. The conditional jump instructions "JC rel" (Jump on Carry) and "JNC rel" (Jump on Not Carry) test the state of the carry flag, branching if it is a one or zero, respectively. (The letters "rel" denote relative code addressing.) The three-byte instructions "JB bit, rel" and "JNB bit, rel" (Jump on Bit and Jump on Not Bit) test the state of any addressable bit in a similar manner. A fifth instruction combines the Jump on Bit and Clear operations. "JBC bit, rel" conditionally branches to the indicated address, then clears the bit in the same two cycle instruction. This operation is the same as the MCS-48™ "JTF" instructions.

All 8051 conditional jump instructions use program counter-relative addressing, and all execute in two cycles. The last instruction byte encodes a signed displacement ranging from -128 to +127. During execution, the CPU adds this value to the incremented program counter to produce the jump destination. Put another way, a conditional jump to the immediately following instruction would encode 00H in the offset byte.

A section of program or subroutine written using only relative jumps to nearby addresses will have the same machine code independent of the code's location. An assembled routine may be repositioned anywhere in memory, even crossing memory page boundaries, without having to modify the program or recompute destination addresses. To facilitate this flexibility, there is an unconditional "Short Jump" (SJMP) which uses relative addressing as well. Since a pro-

programmer would have quite a chore trying to compute relative offset values from one instruction to another. ASM51 automatically computes the displacement needed given only the destination address or label. An error message will alert the programmer if the destination is "out of range."

(The so-called "Bit Test" instructions implemented on many other microprocessors simply perform the logical-AND operation between a byte variable and a constant mask, and set or clear a zero flag depending on the result. This is essentially equivalent to the 8051 "MOV C,bit" instruction. A second instruction is then needed to conditionally branch based on the state of the zero flag. This does *not* constitute abstract bit-addressing in the MCS-51™ sense. A flag exists only as a field within a register; to reference a bit the programmer must know and specify both the encompassing register and the bit's position therein. This constraint severely limits the flexibility of symbolic bit addressing and reduces the machine's code-efficiency and speed.)

Interaction with Other Instructions. The carry flag is also affected by the instructions listed in Table 3. It can be rotated through the accumulator, and altered as a side effect of arithmetic instructions. Refer to the User's Manual for details on how these instructions operate.

Simple Instruction Combinations

By combining general purpose bit operations with certain addressable bits, one can "custom build" several hundred useful instructions. All eight bits of the PSW can be tested directly with conditional jump instructions to monitor (among other things) parity and overflow status. Programmers can take advantage of 128 software flags to keep track of operating modes, resource usage, and so forth.

The Boolean instructions are also the most efficient way to control or reconfigure peripheral and I/O registers. All 32 I/O lines become "test pins," for example, tested by conditional jump instructions. Any output pin can be toggled (complemented) in a single instruction cycle. Setting or clearing the Timer Run flags (TR0 and TR1) turn the timer/counters on or off; polling the same flags elsewhere lets the program determine if a timer is running. The respective overflow flags (TF0 and TF1) can be tested to determine when the desired period or count has elapsed, then cleared in preparation for the next repetition. (For the record, these bits are all part of the TCON register, Figure 7.a. Thanks to symbolic bit addressing, the programmer only needs to remember the mnemonic associated with each function. In other words, don't bother memorizing control word layouts.)

In the MCS-48® family, instructions corresponding to some of the above functions require specific opcodes. Ten different opcodes serve to clear/complement the software flags F0 and F1, enable/disable each interrupt, and start/stop the timer. In the 8051 instruction set, just three opcodes (SETB,

Table 3. Other Instructions Affecting the Carry Flag.

Mnemonic	Description	Byte	Cyc
ADD A,Rn	Add register to Accumulator	1	1
ADD A,direct	Add direct byte to Accumulator	2	1
ADD A,@Ri	Add indirect RAM to Accumulator	1	1
ADD A,#data	Add immediate data to Accumulator	2	1
ADDC A,Rn	Add register to Accumulator with Carry flag	1	1
ADDC A,direct	Add direct byte to Accumulator with Carry flag	2	1
ADDC A,@Ri	Add indirect RAM to Accumulator with Carry flag	1	1
ADDC A,#data	Add immediate data to Acc with Carry flag	2	1
SUBB A,Rn	Subtract register from Accumulator with borrow	1	1
SUBB A,direct	Subtract direct byte from Acc with borrow	2	1
SUBB A,@Ri	Subtract indirect RAM from Acc with borrow	1	1
SUBB A,#data	Subtract immediate data from Acc with borrow	2	1
MUL AB	Multiply A & B	1	4
DIV AB	Divide A by B	1	4
DA A	Decimal Adjust Accumulator	1	1
RLC A	Rotate Accumulator Left through the Carry flag	1	1
RRC A	Rotate Accumulator Right through Carry flag	1	1
CJNE A,direct,rel	Compare direct byte to Acc & Jump if Not Equal	3	2
CJNE A,#data,rel	Compare immediate to Acc & Jump if Not Equal	3	2
CJNE Rn,#data,rel	Compare immed to register & Jump if Not Equal	3	2
CJNE @Ri,#data,rel	Compare immed to indirect & Jump if Not Equal	3	2

All mnemonics copyrighted © Intel Corporation 1980.

CLR, CPL) with a direct bit address appended perform the same functions. Two test instructions (JB and JNB) can be combined with bit addresses to test the software flags, the 8048 I/O pins T0, T1, and INT, and the eight accumulator bits, replacing 15 more different instructions.

Table 4.a shows how 8051 programs implement software flag and machine control functions associated with special

using awkward sequences of other basic operations. As mentioned earlier, any CPU can solve any problem given enough time.

Quantitatively, the differences between a solution allowed by the 8051 and those required by previous architectures are numerous. What the 8051 Family buys you is a faster, cleaner, lower-cost solution to microcontroller applications.

The opcode space freed by condensing many specific 8048

Table 4.a. Contrasting 8048 and 8051 Bit Control and Testing Instructions.

8048 Instruction	Bytes	Cycles	uSec	8x51 Instruction	Bytes	Cycles & uSec
Flag Control						
CLR C	1	1	2.5	CLR C	1	1
CPL F0	1	1	2.5	CPL F0	2	1
Flag Testing						
JNC offset	2	2	5.0	JNC rel	2	2
JF0 offset	2	2	5.0	JB F0,rel	3	2
JB7 offset	2	2	5.0	JB ACC.7,rel	3	2
Peripheral Polling						
JT0 offset	2	2	5.0	JB T0,rel	3	2
JN1 offset	2	2	5.0	JNB INT0,rel	3	2
JTF offset	2	2	5.0	JBC TF0,rel	3	2
Machine and Peripheral Control						
STRT T	1	1	2.5	SETB TR0	2	1
EN I	1	1	2.5	SETB EX0	2	1
DIS TCNT1	1	1	2.5	CLR ET0	2	1

Table 4.b. Replacing 8048 instruction sequences with single 8x51 instructions.

8048 Instructions	Bytes	Cycles	uSec	8051 Instructions	Bytes	Cycles & uSec
Flag Control						
Set carry:						
CLR C	1	1	2.5	SETB C	1	1
CPL C	1	1	2.5			
Set Software Flag:						
CLR F0	1	1	2.5	SETB F0	2	1
CPL F0	1	1	2.5			

opcodes in the 8048. In every case the MCS-51™ solution requires the same number of machine cycles, and executes 2.5 times faster.

3. BOOLEAN PROCESSOR APPLICATIONS

So what? Then what does all this buy you?

Qualitatively, nothing. All the same capabilities *could* be (and often have been) implemented on other machines

instructions into a few general operations has been used to add new functionality to the MCS-51™ architecture - both for byte and bit operations. 144 software flags replace the 8048's two. These flags (and the carry) may be directly set, not just cleared and complemented, and all can be tested for either state, not just one. Operating mode bits previously inaccessible may be read, tested, or saved. Situations where the 8051 instruction set provides new capabilities are contrasted with 8048 instruction sequences in Table 4.b. Here the 8051 speed advantage ranges from 5x to 15x!

8048 Instructions	Bytes	Cycles	uSec	8x51 Instructions	Bytes	Cycles & uSec
Turn Off Output Pin: ANL P1,#0FBH	2	2	5.0	CLR P1.2	2	2
Complement Output Pin: IN A,P1 XRL A,#04H OUTL P1,A	4	6	15.0	CPL P1.2	2	1
Clear Flag in RAM: MOV R0,#FLGADR MOV A,@R0 ANL A,#FLGMASK MOV @R0,A	6	6	15.0	CLR USER_FLG	2	1
Flag Testing Jump if Software Flag is 0: JF0 \$+4 JMP offset	4	4	10.0	JNB F0,rel	3	2
Jump if Accumulator bit is 0: CPL A JB7 offset CPL A	4	4	10.0	JNB ACC.7,rel	3	2
Peripheral Polling Test if Input Pin is Grounded: IN A,P1 CPL A JB3 offset	4	5	12.5	JNB P1.3,rel	3	2
Test if Interrupt Pin is High: JN1 \$+4 JMP offset	4	4	10.0	JB INT0,rel	3	2

Combining Boolean and byte-wide instructions can produce great synergy. An MCS-51™ based application will prove to be:

- simpler to write since the architecture correlates more closely with the problems being solved;
- easier to debug because more individual instructions have no unexpected or undesirable side-effects;
- more byte efficient due to direct bit addressing and program counter relative branching;
- faster running because fewer bytes of instruction need to be fetched and fewer conditional jumps are processed;
- lower cost because of the high level of system-integration within one component.

These rather unabashed claims of excellence shall not go unsubstantiated. The rest of this chapter examines less trivial tasks simplified by the Boolean processor. The first

three compare the 8051 with other microprocessors; the last two go into 8051-based system designs in much greater depth.

Design Example #1 - Bit Permutation

First off, we'll use the bit-transfer instructions to permute a lengthy pattern of bits.

A steadily increasing number of data communication products use encoding methods to protect the security of sensitive information. By law, interstate financial transactions involving the Federal banking system must be transmitted using the Federal Information Processing *Data Encryption Standard* (DES).

Basically, the DES combines eight bytes of "plaintext" data (in binary, ASCII, or any other format) with a 56-bit "key", producing a 64-bit encrypted value for transmission. At the receiving end the same algorithm is applied to the incoming data using the same key, reproducing the original eight byte message. The algorithm used for these permutations is fixed; different user-defined keys ensure data privacy.

It is not the purpose of this note to describe the DES in any detail. Suffice it to say that encryption/decryption is a long, iterative process consisting of rotations, exclusive-OR operations, function table look-ups, and an extensive (and quite bizarre) sequence of bit permutation, packing, and unpacking steps. (For further details refer to the June 21, 1979 issue of *Electronics* magazine.) The bit manipulation steps are included, it is rumored, to impede a general purpose digital supercomputer trying to "break" the code. Any algorithm implementing the DES with previous generation microprocessors would spend virtually all of its time diddling bits.

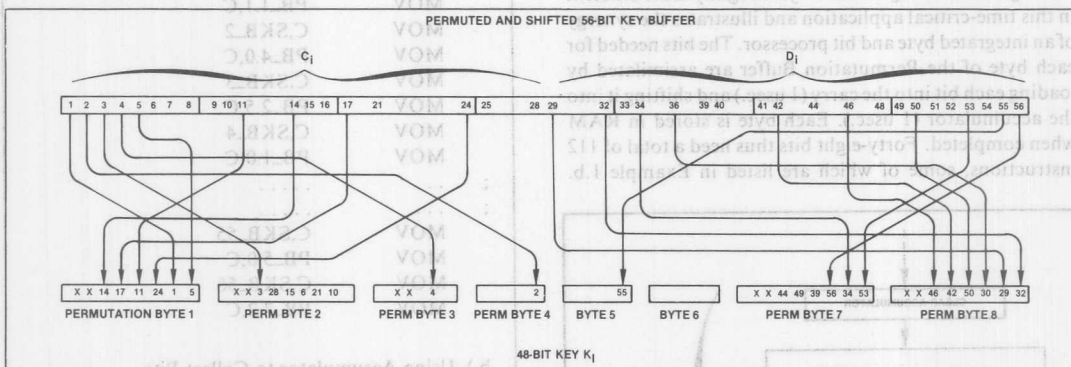


Figure 9. DES Key Schedule Transformation.

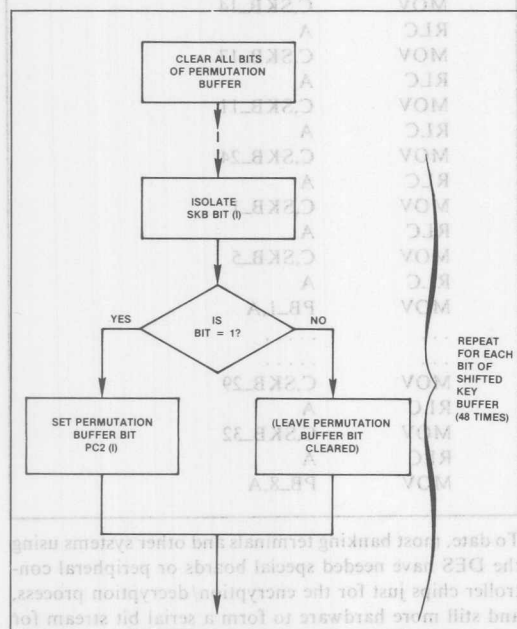


Figure 10.a. Flowchart for Key permutation attempted with a byte processor.

The bit manipulation performed is typified by the Key Schedule Calculation represented in Figure 9. This step is repeated 16 times for each key used in the course of a transmission. In essence, a seven-byte, 56-bit "Shifted Key Buffer" is transformed into an eight-byte, "Permutation Buffer" without altering the shifted Key. The arrows in Figure 9 indicate a few of the translation steps. Only six bits of each byte of the Permutation Buffer are used; the two high-order bits of each byte are cleared. This means only 48 of the 56 Shifted Key Buffer bits are used in any one iteration.

Different microprocessor architectures would best implement this type of permutation in different ways. Most approaches would share the steps of Figure 10.a:

- Initialize the Permutation Buffer to default state (ones or zeroes);
- Isolate the state of a bit of a byte from the Key Buffer. Depending on the CPU, this might be accomplished by rotating a word of the Key Buffer through a carry flag or testing a bit in memory or an accumulator against a mask byte;
- Perform a conditional jump based on the carry or zero flag if the Permutation Buffer default state is correct;
- Otherwise reverse the corresponding bit in the permutation buffer with logical operations and mask bytes.

Each step above may require several instructions. The last three steps must be repeated for all 48 bits. Most microprocessors would spend 300 to 3,000 microseconds on each of the 16 iterations.

Notice, though, that this flow chart looks a lot like Figure 8. The Boolean Processor can permute bits by simply moving

them from the source to the carry to the destination—a total of two instructions taking four bytes and three microseconds per bit. Assume the Shifted Key Buffer and Permutation Buffer both reside in bit-addressable RAM, with the bits of the former assigned symbolic names SKB_1, SKB_2, ..., SKB_56, and that the bytes of the latter are named PB_1, ..., PB_8. Then working from Figure 9, the software for the permutation algorithm would be that of Example 1.a. The total routine length would be 192 bytes, requiring 144 microseconds.

The algorithm of Figure 10.b is just slightly more efficient in this time-critical application and illustrates the synergy of an integrated byte and bit processor. The bits needed for each byte of the Permutation Buffer are assimilated by loading each bit into the carry (1 usec.) and shifting it into the accumulator (1 usec.). Each byte is stored in RAM when completed. Forty-eight bits thus need a total of 112 instructions, some of which are listed in Example 1.b.

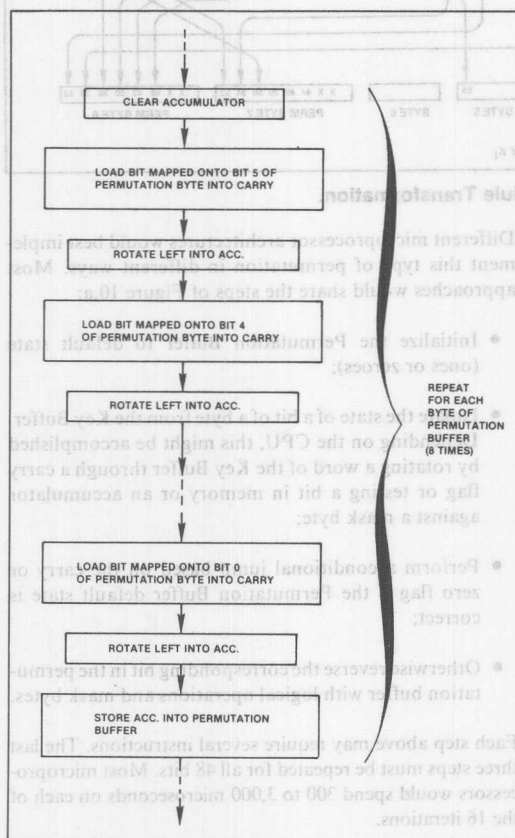


Figure 10.b. DES Key Permutation with Boolean Processor.

Worst-case execution time would be 112 microseconds, since each instruction takes a single cycle. Routine length would also decrease, to 168 bytes. (Actually, in the context of the complete encryption algorithm, each permuted byte would be processed as soon as it is assimilated—saving memory and cutting execution time by another 8 usec.)

Example 1. DES Key Permutation Software.

a.) "Brute Force" technique.

```

MOV    C,SKB_1
MOV    PB_1.1,C
MOV    C,SKB_2
MOV    PB_4.0,C
MOV    C,SKB_3
MOV    PB_2.5,C
MOV    C,SKB_4
MOV    PB_1.0,C
...
MOV    C,SKB_55
MOV    PB_5.0,C
MOV    C,SKB_56
MOV    PB_7.2,C
  
```

b.) Using Accumulator to Collect Bits.

```

CLR    A
MOV    C,SKB_14
RLC    A
MOV    C,SKB_17
RLC    A
MOV    C,SKB_11
RLC    A
MOV    C,SKB_24
RLC    A
MOV    C,SKB_1
RLC    A
MOV    C,SKB_5
RLC    A
MOV    PB_1,A
...
MOV    C,SKB_29
RLC    A
MOV    C,SKB_32
RLC    A
MOV    PB_8,A
  
```

To date, most banking terminals and other systems using the DES have needed special boards or peripheral controller chips just for the encryption/decryption process, and still more hardware to form a serial bit stream for transmission (Figure 11.a). An 8051 solution could pack most of the entire system onto the one chip (Figure 11.b). The whole DES algorithm would require less than one-

fourth of the on-chip program memory, with the remaining bytes free for operating the banking terminal (or whatever) itself.

Moreover, since transmission and reception of data is performed through the on-board UART, the unencrypted data (plaintext) never even exists outside the microcomputer! Naturally, this would afford a high degree of security from data interception.

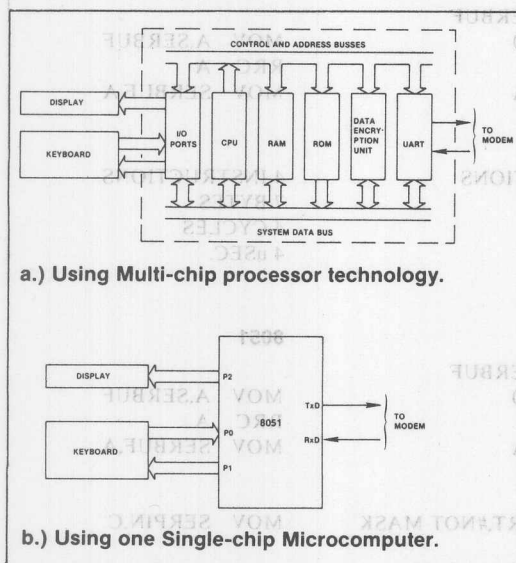


Figure 11. Secure Banking Terminal Block Diagram.

Design Example #2 - Software Serial I/O

An exercise often imposed on beginning microcomputer students is to write a program simulating a UART. (See, for example, Application Notes AP24, AP29, and AP49.) Though doing this with the 8051 Family may appear to be a moot point (given that the hardware for a full UART is on-chip), it is still instructive to see how it would be done, and maintains a product line tradition.

As it turns out, the 8051 microcomputers can receive or transmit serial data via software very efficiently using the Boolean instruction set. Since any I/O pin may be a serial input or output, several serial links could be maintained at once.

Figures 12.a and 12.b show algorithms for receiving or transmitting a byte of data. (Another section of program would invoke this algorithm eight times, synchronizing it with a start bit, clock signal, software delay, or timer

interrupt.) Data is received by testing an input pin, setting the carry to the same state, shifting the carry into a data buffer, and saving the partial frame in internal RAM. Data is transmitted by shifting an output buffer through the carry, and generating each bit on an output pin.

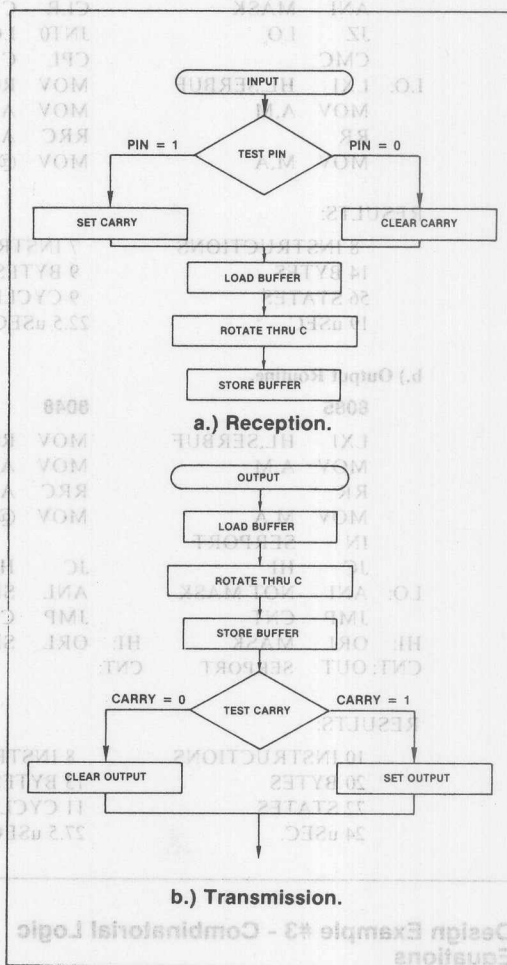


Figure 12. Serial I/O Algorithms.

A side-by-side comparison of the software for this common "bit-banging" application with three different microprocessor architectures is shown in Table 5.a and 5.b. The 8051 solution is more efficient than the others on every count!

**Table 5. Serial I/O Programs
for Various Microprocessors.**

<p>a.) Input Routine.</p> <p>8085</p> <pre> IN SERPORT ANI MASK JZ LO CMC LO: LXI HL,SERBUF MOV A,M RR MOV M,A </pre> <p>RESULTS:</p> <p>8 INSTRUCTIONS 14 BYTES 56 STATES 19 uSEC.</p>	<p>8048</p> <pre> CLR C JNT0 LO CPL C MOV R0,#SERBUF MOV A,@R0 RR A MOV @R0,A </pre> <p>7 INSTRUCTIONS 9 BYTES 9 CYCLES 22.5 uSEC.</p>	<p>8051</p> <pre> MOV C,SERPIN MOV A,SERBUF RRC A MOV SERBUF,A </pre> <p>4 INSTRUCTIONS 7 BYTES 4 CYCLES 4 uSEC.</p>
<p>b.) Output Routine.</p> <p>8085</p> <pre> LXI HL,SERBUF MOV A,M RR MOV M,A IN SERPORT JC HI LO: ANI NOT MASK JMP CNT HI: ORI MASK CNT: OUT SERPORT </pre> <p>RESULTS:</p> <p>10 INSTRUCTIONS 20 BYTES 72 STATES 24 uSEC.</p>	<p>8048</p> <pre> MOV R0,#SERBUF MOV A,@R0 RRC A MOV @R0,A JC HI ANL SERPRT,#NOT MASK JMP CNT HI: ORL SERPRT,#MASK CNT: </pre> <p>8 INSTRUCTIONS 13 BYTES 11 CYCLES 27.5 uSEC.</p>	<p>8051</p> <pre> MOV A,SERBUF RRC A MOV SERBUF,A MOV SERPIN,C </pre> <p>4 INSTRUCTIONS 7 BYTES 5 CYCLES 5 uSEC.</p>

Design Example #3 - Combinatorial Logic Equations

Next we'll look at some simple uses for bit-test instructions and logical operations. (This example is also presented in Application Note AP-69.)

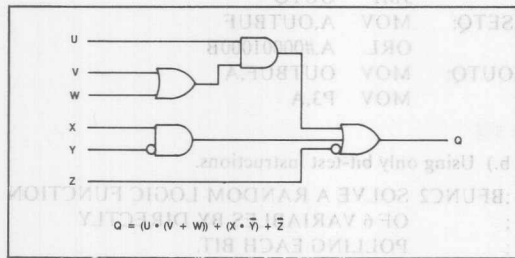
Virtually all hardware designers have solved complex functions using combinatorial logic. While the hardware involved may vary from relay logic, vacuum tubes, or TTL or to more esoteric technologies like fluidics, in each case the goal is the same: to solve a problem represented by a logical function of several Boolean variables.

Figure 13 shows TTL and relay logic diagrams for a function of the six variables U through Z. Each is a solution of the equation.

$$Q = (U \vee (V \vee W)) + (X \vee \bar{Y}) + \bar{Z}$$

Equations of this sort might be reduced using Karnaugh Maps or algebraic techniques, but that is not the purpose of this example. As the logic complexity increases, so does the difficulty of the reduction process. Even a minor change to the function equations as the design evolves would require tedious re-reduction from scratch.

Figure 13. Hardware Implementations of Boolean functions.



a.) Using TTL:

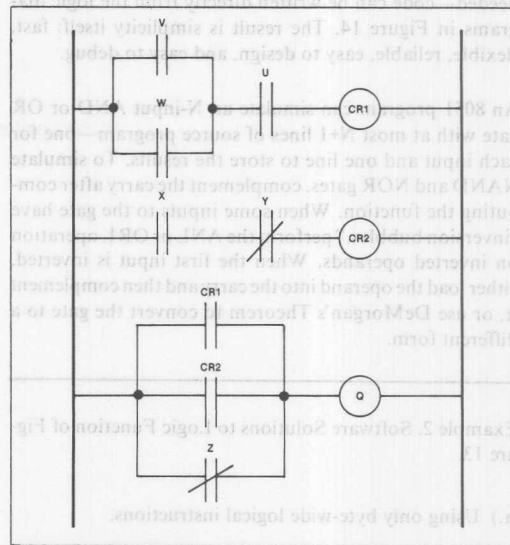
For the sake of comparison we will implement this function three ways, restricting the software to three proper subsets of the MCS-51™ instruction set. We will also assume that U and V are input pins from different input ports, W and X are status bits for two peripheral controllers, and Y and Z are software flags set up earlier in the program. The end result must be written to an output pin on some third port. The first two implementations follow the flow-chart shown in Figure 14. Program flow would embark on a route down a test-and-branch tree and leaves either the "True" or "Not True" exit ASAP — as soon as the proper result has been determined. These exits then rewrite the output port with the result bit respectively one or zero.

Other digital computers must solve equations of this type with standard word-wide logical instructions and conditional jumps. So for the first implementation, we won't use any generalized bit-addressing instructions. As we shall soon see, being constrained to such an instruction subset produces somewhat sloppy software solutions. MCS-51™ mnemonics are used in Example 2.a; other machines might further cloud the situation by requiring operation-specific mnemonics like INPUT, OUTPUT, LOAD, STORE, etc., instead of the MOV mnemonic used for all variable transfers in the 8051 instruction set.

The code which results is cumbersome and error prone. It would be difficult to prove whether the software worked for all input combinations in programs of this sort. Furthermore, execution time will vary widely with input data.

Thanks to the direct bit-test operations, a single instruction can replace each move/mask/conditional jump sequence in Example 2.a, but the algorithm would be equally convoluted (see Example 2.B). To lessen the confusion "a bit" each input variable is assigned a symbolic name.

A more elegant and efficient implementation (Example 2.c) strings together the Boolean AND and ORL functions to generate the output function with straight-line code.



b.) Using Relay Logic:

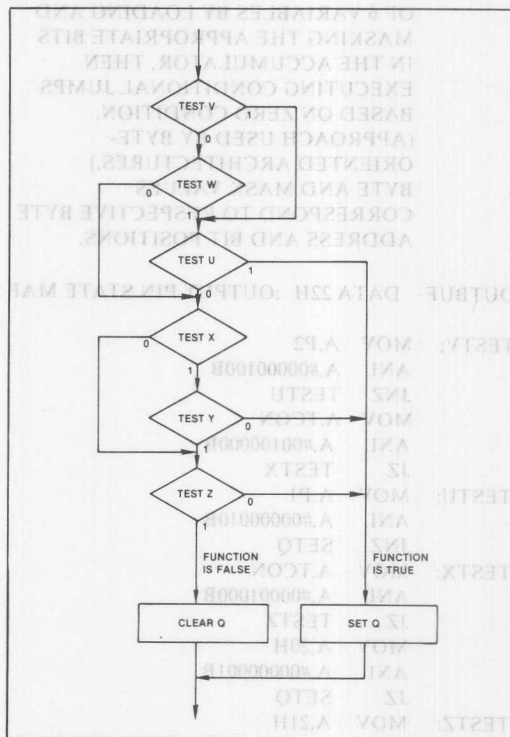
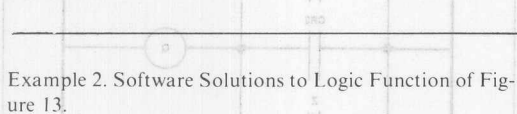


Figure 14. Flow chart for tree-branching algorithm.

needed—code can be written directly from the logic diagrams in Figure 14. The result is simplicity itself: fast, flexible, reliable, easy to design, and easy to debug.

An 8051 program can simulate an N-input AND or OR gate with at most N+1 lines of source program—one for each input and one line to store the results. To simulate NAND and NOR gates, complement the carry after computing the function. When some inputs to the gate have “inversion bubbles,” perform the ANL or ORL operation on inverted operands. When the first input is inverted, either load the operand into the carry and then complement it, or use DeMorgan’s Theorem to convert the gate to a different form.



Example 2. Software Solutions to Logic Function of Figure 13.

a.) Using only byte-wide logical instructions.

```

:BFUNC1 SOLVE RANDOM LOGIC FUNCTION
: OF 6 VARIABLES BY LOADING AND
: MASKING THE APPROPRIATE BITS
: IN THE ACCUMULATOR, THEN
: EXECUTING CONDITIONAL JUMPS
: BASED ON ZERO CONDITION.
: (APPROACH USED BY BYTE-
: ORIENTED ARCHITECTURES.)
: BYTE AND MASK VALUES
: CORRESPOND TO RESPECTIVE BYTE
: ADDRESS AND BIT POSITIONS.
:

```

OUTBUF DATA 22H :OUTPUT PIN STATE MAP

```

TESTV: MOV A,P2
      ANL A,#0000100B
      JNZ TESTU
      MOV A,TCON
      ANL A,#00100000B
      JZ TESTX
TESTU: MOV A,P1
      ANL A,#00000010B
      JNZ SETQ
TESTX: MOV A,TCON
      ANL A,#00001000B
      JZ TESTZ
      MOV A,20H
      ANL A,#00000001B
      JZ SETQ
TESTZ: MOV A,21H
      ANL A,#00000010B
      JZ SETQ

```

```

      JMP OUTQ
SETQ: MOV A,OUTBUF
      ORL A,#00001000B
OUTQ: MOV OUTBUF,A
      MOV P3,A

```

b.) Using only bit-test instructions.

```

:BFUNC2 SOLVE A RANDOM LOGIC FUNCTION
: OF 6 VARIABLES BY DIRECTLY
: POLLING EACH BIT.
:
: (APPROACH USING MCS-51 UNIQUE
: BIT-TEST INSTRUCTION CAPABILITY.)
: SYMBOLS USED IN LOGIC DIAGRAM
: ASSIGNED TO CORRESPONDING 8x51
: BIT ADDRESSES.
:

```

```

      BIT P1.1
      BIT P2.2
      BIT TF0
      BIT IE1
      BIT 20H.0
      BIT 21H.1
      BIT P3.3
TEST_V: JB V.TEST_U
      JNB W.TEST_X
TEST_U: JB U.SET_Q
TEST_X: JNB X.TEST_Z
      JNB Y.SET_Q
TEST_Z: JNB Z.SET_Q
CLR_Q: CLR Q
      JMP NXTTST
SET_Q: SETB Q
NXTTST: ;(CONTINUATION OF
:PROGRAM)

```

c.) Using logical operations on Boolean variables.

```

:FUNC3 SOLVE A RANDOM LOGIC FUNCTION
: OF 6 VARIABLES USING
: STRAIGHT-LINE LOGICAL
: INSTRUCTIONS ON MCS-51 BOOLEAN
: VARIABLES.
:

```

```

      MOV C,V
      ORL C,W :OUTPUT OF OR GATE
      ANL C,U :OUTPUT OF TOP AND GATE
      MOV F0,C :SAVE INTERMEDIATE STATE
      MOV C,X
      ANL C,Y :OUTPUT OF BOTTOM AND GATE
      ORL C,F0 :INCLUDE VALUE SAVED ABOVE
      ORL C,Z :INCLUDE LAST INPUT VARIABLE
      MOV Q,C :OUTPUT COMPUTED RESULT

```


An upper-limit can be placed on the complexity of software to simulate a large number of gates by summing the total number of inputs and outputs. The *actual* total should be somewhat shorter, since calculations can be "chained," as shown above. The output of one gate is often the first input to another, bypassing the intermediate variable to eliminate two lines of source.

Design Example #4 - Automotive Dashboard Functions

Now let's apply these techniques to designing the software for a complete controller system. This application is patterned after a familiar real-world application which isn't nearly as trivial as it might first appear: automobile turn signals.

Imagine the three position turn lever on the steering column as a single-pole, triple-throw toggle switch. In its central position all contacts are open. In the up or down positions contacts close causing corresponding lights in the rear of the car to blink. So far very simple.

Two more turn signals blink in the front of the car, and two others in the dashboard. All six bulbs flash when an emergency switch is closed. A thermo-mechanical relay (accessible under the dashboard in case it wears out) causes the blinking.

Applying the brake pedal turns the tail light filaments on constantly . . . unless a turn is in progress, in which case the blinking tail light is not affected. (Of course, the front turn signals and dashboard indicators are not affected by the brake pedal.) Table 6 summarizes these operating modes.

But we're not done yet. Each of the exterior turn signal (but not the dashboard) bulbs has a second, somewhat dimmer filament for the parking lights. Figure 15 shows TTL circuitry which could control all six bulbs. The signals labeled "High Freq." and "Low Freq." represent two square-wave inputs. Basically, when one of the turn switches is closed or the emergency switch is activated the low frequency signal (about 1 Hz) is gated through to the appropriate dashboard indicator(s) and turn signal(s). The rear signals are also activated when the brake pedal is depressed provided a turn is not being made in the same direction. When the parking light switch is closed the higher frequency oscillator is gated to each front and rear turn signal, sustaining a low-intensity background level. (This is to eliminate the need for additional parking light filaments.)

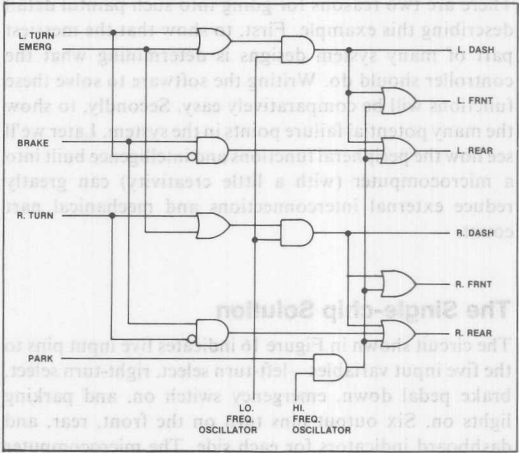


Figure 15. TTL logic implementation of automotive turn signals.

Table 6. Truth table for turn-signal operation.

INPUT SIGNALS				OUTPUT SIGNALS			
BRAKE SWITCH	EMERG. SWITCH	LEFT TURN SWITCH	RIGHT TURN SWITCH	LEFT FRONT & DASH	RIGHT FRONT & DASH	LEFT REAR	RIGHT REAR
0	0	0	0	OFF	OFF	OFF	OFF
0	0	0	1	OFF	BLINK	OFF	BLINK
0	0	1	0	BLINK	OFF	BLINK	OFF
0	1	0	0	BLINK	BLINK	BLINK	BLINK
0	1	0	1	BLINK	BLINK	BLINK	BLINK
0	1	1	0	BLINK	BLINK	BLINK	BLINK
1	0	0	0	OFF	OFF	ON	ON
1	0	0	1	OFF	BLINK	ON	BLINK
1	0	1	0	BLINK	OFF	BLINK	ON
1	1	0	0	BLINK	BLINK	ON	ON
1	1	0	1	BLINK	BLINK	ON	BLINK
1	1	1	0	BLINK	BLINK	BLINK	ON

In most cars, the switching logic to generate these functions requires a number of multiple-throw contacts. As many as 18 conductors thread the steering column of some automobiles solely for turn-signal and emergency blinker functions. (The author discovered this recently to his astonishment and dismay when replacing the whole assembly because of one burned contact.)

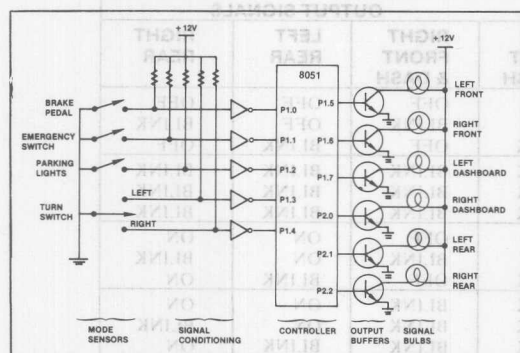
A multiple-conductor wiring harness runs to each corner of the car, behind the dash, up the steering column, and down to the blinker relay below. Connectors at each termination for each filament lead to extra cost and labor during construction, lower reliability and safety, and more costly repairs. And considering the system's present complexity, increasing its reliability or detecting failures would be quite difficult.

There are two reasons for going into such painful detail describing this example. First, to show that the messiest part of many system designs is determining what the controller should do. Writing the software to solve these functions will be comparatively easy. Secondly, to show the many potential failure points in the system. Later we'll see how the peripheral functions and intelligence built into a microcomputer (with a little creativity) can greatly reduce external interconnections and mechanical part count.

The Single-chip Solution

The circuit shown in Figure 16 indicates five input pins to the five input variables—left-turn select, right-turn select, brake pedal down, emergency switch on, and parking lights on. Six output pins turn on the front, rear, and dashboard indicators for each side. The microcomputer implements all logical functions through software, which periodically updates the output signals as time elapses and input conditions change.

Figure 16. Microcomputer Turn-signal Connections.



Design Example #3 demonstrated that symbolic addressing with user-defined bit names makes code and documentation easier to write and maintain. Accordingly, we'll assign these I/O pins names for use throughout the program. (The format of this example will differ somewhat from the others. Segments of the overall program will be presented in sequence as each is described.)

```

:
: INPUT PIN DECLARATIONS:
: (ALL INPUTS ARE POSITIVE-TRUE LOGIC)
:
BRAKE BIT P1.0 : BRAKE PEDAL DEPRESSED
EMERG BIT P1.1 : EMERGENCY BLINKER
                ACTIVATED
PARK BIT P1.2 : PARKING LIGHTS ON
L_TURN BIT P1.3 : TURN LEVER DOWN
R_TURN BIT P1.4 : TURN LEVER UP
:
: OUTPUT PIN DECLARATIONS:
:
L_FRNT BIT P1.5 : FRONT LEFT-TURN
                INDICATOR
R_FRNT BIT P1.6 : FRONT RIGHT-TURN
                INDICATOR
L_DASH BIT P1.7 : DASHBOARD LEFT-TURN
                INDICATOR
R_DASH BIT P2.0 : DASHBOARD RIGHT-TURN
                INDICATOR
L_REAR BIT P2.1 : REAR LEFT-TURN
                INDICATOR
R_REAR BIT P2.2 : REAR RIGHT-TURN
                INDICATOR

```

Another key advantage of symbolic addressing will appear further on in the design cycle. The locations of cable connectors, signal conditioning circuitry, voltage regulators, heat sinks, and the like all affect P.C. board layout. It's quite likely that the somewhat arbitrary pin assignment defined early in the software design cycle will prove to be less than optimum; rearranging the I/O pin assignment could well allow a more compact module, or eliminate costly jumpers on a single-sided board. (These considerations apply especially to automotive and other cost-sensitive applications needing single-chip controllers.) Since other architectures mask bytes or use "clever" algorithms to isolate bits by rotating them into the carry, re-routing an input signal (from bit 1 of port 1, for example, to bit 4 of port 3) could require extensive modifications throughout the software.

The Boolean Processor's direct bit addressing makes such changes absolutely trivial. The number of the port containing the pin is irrelevant, and masks and complex program structures are not needed. Only the initial Boolean varia-

```

SUB_DIV DATA 20H
: HIGH-FREQUENCY OSCILLATOR BIT
HL_FREQ BIT SUB_DIV.0
: LOW-FREQUENCY OSCILLATOR BIT
LO_FREQ BIT SUB_DIV.7
:
: ORG 0000H
JMP INIT
:
: ORG 100H
: PUT TIMER 0 IN MODE 1
INIT: MOV TMOD,#00000001B
: INITIALIZE TIMER REGISTERS
MOV TL0,#0
MOV TH0,#-16
: SUBDIVIDE INTERRUPT RATE BY 244
MOV SUB_DIV,#244
: ENABLE TIMER INTERRUPTS
SETB ET0
: GLOBALLY ENABLE ALL INTERRUPTS
SETB EA
: START TIMER
SETB TR0
: (CONTINUE WITH BACKGROUND PROGRAM)
: PUT TIMER 0 IN MODE 1
: INITIALIZE TIMER REGISTERS
: SUBDIVIDE INTERRUPT RATE BY 244
: ENABLE TIMER INTERRUPTS
: GLOBALLY ENABLE ALL INTERRUPTS
: START TIMER

```

ble declarations need to be changed; ASM51 automatically adjusts all addresses and symbolic references to the reassigned variables. The user is assured that no additional debugging or software verification will be required.

Timer 0 (one of the two on-chip timer/counters) replaces the thermo-mechanical blinker relay in the dashboard controller. During system initialization it is configured as a timer in mode 1 by setting the least significant bit of the timer mode register (TMOD). In this configuration the low-order byte (TL0) is incremented every machine cycle, overflowing and incrementing the high-order byte (TH0) every 256 μ Sec. Timer interrupt 0 is enabled so that a hardware interrupt will occur each time TH0 overflows. (For details of the numerous timer operating modes see the MCS-51™ User's Manual.)

An eight-bit variable in the bit-addressable RAM array will be needed to further subdivide the interrupts via software. The lowest-order bit of this counter toggles very

indicator blinking rate.

Loading TH0 with -16 will cause an interrupt after 4.096 msec. The interrupt service routine reloads the high-order byte of timer 0 for the next interval, saves the CPU registers likely to be affected on the stack, and then decrements SUB_DIV. Loading SUB_DIV with 244 initially and each time it decrements to zero will produce a 0.999 second period for the highest-order bit.

```

ORG 000BH : TIMER 0 SERVICE VECTOR
MOV TH0,#-16
PUSH PSW
PUSH ACC
PUSH B
DJNZ SUB_DIV,T0SERV
MOV SUB_DIV,#244

```

The code to sample inputs, perform calculations, and update outputs—the real “meat” of the signal controller algorithm—may be performed either as part of the interrupt service routine or as part of a background program loop. The only concern is that it must be executed at least several dozen times per second to prevent parking light flickering. We will assume the former case, and insert the code into the timer 0 service routine.

First, notice from the logic diagram (Figure 15) that the subterm (PARK · H_FREQ), asserted when the parking lights are to be on dimly, figures into four of the six output functions. Accordingly, we will first compute that term and save it in a temporary location named “DIM”. The PSW contains two general purpose flags: F0, which corresponds to the 8048 flag of the same name, and PSW.1. Since The PSW has been saved and will be restored to its previous state after servicing the interrupt, we can use either bit for temporary storage.

```

DIM BIT PSW.1 : DECLARE TEMP.
                STORAGE FLAG
:
: MOV C,PARK : GATE PARKING
: ANL HL,FREQ : LIGHT SWITCH
:              : WITH HIGH
:              : FREQUENCY
:              : SIGNAL
MOV DIM,C : AND SAVE IN
:              : TEMP. VARIABLE.

```

This simple three-line section of code illustrates a remarkable point. The software indicates in very abstract terms exactly what function is being performed, independent of

include an input pin, a bit within a program variable, and a software flag in the PSW is totally invisible to the programmer.

Now generate and output the dashboard left turn signal.

```
MOV C, L_TURN : SET CARRY IF
                TURN
ORL C, EMERG   : OR EMERGENCY
                SELECTED.
ANL C, LO_FREQ : GATE IN 1 HZ
                SIGNAL.
MOV L_DASH.C   : AND OUTPUT TO
                DASHBOARD.
```

To generate the left front turn signal we only need to add the parking light function in F0. But notice that the function in the carry will also be needed for the rear signal. We can save effort later by saving its current state in F0.

```
MOV F0.C       : SAVE FUNCTION
                SO FAR.
ORL C, DIM      : ADD IN PARKING
                LIGHT FUNCTION
MOV L_FRNT.C    : AND OUTPUT TO
                TURN SIGNAL.
```

Finally, the rear left turn signal should also be on when the brake pedal is depressed, provided a left turn is not in progress.

```
MOV C, BRAKE   : GATE BRAKE
                PEDAL SWITCH
ANL C, L_TURN   : WITH TURN
                LEVER.
ORL C, F0       : INCLUDE TEMP.
                VARIABLE FROM
                DASH.
ORL C, DIM      : AND PARKING
                LIGHT FUNCTION
MOV L_REAR.C    : AND OUTPUT TO
                TURN SIGNAL.
```

Now we have to go through a similar sequence for the right-hand equivalents to all the left-turn lights. This also gives us a chance to see how the code segments above look when combined.

```
MOV C, R_TURN   : SET CARRY IF
                TURN
ORL C, EMERG     : OR EMERGENCY
                SELECTED.
ANL C, LO_FREQ   : IF SO, GATE IN 1
                HZ SIGNAL
```

```
MOV F0.C        : DASHBOARD,
                : SAVE FUNCTION
                : SO FAR.
ORL C, DIM       : ADD IN PARKING
                : LIGHT FUNCTION
MOV R_FRNT.C     : AND OUTPUT TO
                : TURN SIGNAL.
MOV C, BRAKE     : GATE BRAKE
                : PEDAL SWITCH
ANL C, R_TURN    : WITH TURN
                : LEVER.
ORL C, F0        : INCLUDE TEMP.
                : VARIABLE FROM
                : DASH
ORL C, DIM       : AND PARKING
                : LIGHT FUNCTION
MOV R_REAR.C     : AND OUTPUT TO
                : TURN SIGNAL.
```

(The perceptive reader may notice that simply rearranging the steps could eliminate one instruction from each sequence.)

Now that all six bulbs are in the proper states, we can return from the interrupt routine, and the program is finished. This code essentially needs to reverse the status saving steps at the beginning of the interrupt.

```
POP B           : RESTORE CPU
                : REGISTERS.
POP ACC
POP PSW
RETI
```

Program Refinements. The luminescence of an incandescent light bulb filament is generally non-linear; the 50% duty cycle of HL_FREQ may not produce the desired intensity. If the application requires, duty cycles of 25%, 75%, etc. are easily achieved by ANDing and ORing in additional low-order bits of SUB_DIV. For example, 30 Hz signals of seven different duty cycles could be produced by considering bits 2—0 as shown in Table 7. The only software change required would be to the code which sets-up variable DIM:

```
MOV C, SUB_DIV.1 : START WITH 50
                : PERCENT
ANL C, SUB_DIV.0 : MASK DOWN TO 25
                : PERCENT
ORL C, SUB_DIV.2 : AND BUILD BACK TO
                : 62 PERCENT
MOV DIM.C        : DUTY CYCLE FOR
                : PARKING LIGHTS.
```


Table 7. Non-trivial Duty Cycles.

SUB_DIV BITS								DUTY CYCLES						
7	6	5	4	3	2	1	0	12.5%	25.0%	37.5%	50.0%	62.5%	75.0%	87.5%
X	X	X	X	X	0	0	0	OFF	OFF	OFF	OFF	OFF	OFF	OFF
X	X	X	X	X	0	0	1	OFF	OFF	OFF	OFF	OFF	OFF	ON
X	X	X	X	X	0	1	0	OFF	OFF	OFF	OFF	OFF	ON	ON
X	X	X	X	X	0	1	1	OFF	OFF	OFF	OFF	ON	ON	ON
X	X	X	X	X	1	0	0	OFF	OFF	OFF	ON	ON	ON	ON
X	X	X	X	X	1	0	1	OFF	OFF	ON	ON	ON	ON	ON
X	X	X	X	X	1	1	0	OFF	ON	ON	ON	ON	ON	ON
X	X	X	X	X	1	1	1	ON	ON	ON	ON	ON	ON	ON

Interconnections increase cost and decrease reliability. The simple buffered pin-per-function circuit in Figure 16 is insufficient when many outputs require higher-than-TTL drive levels. A lower-cost solution uses the 8051 serial port in the shift-register mode to augment I/O. In mode 0, writing a byte to the serial port data buffer (SBUF) causes the data to be output sequentially through the "RXD" pin while a burst of eight clock pulses is generated on the "TXD" pin. A shift register connected to these pins (Figure 17) will load the data byte as it is shifted out. A number of special peripheral driver circuits combining shift-register inputs with high drive level outputs have been introduced recently.

Cascading multiple shift registers end-to-end will expand the number of outputs even further. The data rate in the I/O expansion mode is one megabaud, or 8 usec. per byte. This is the mode which the serial port defaults to following a reset, so no initialization is required.

The software for this technique uses the B register as a "map" corresponding to the different output functions. The program manipulates these bits instead of the output pins. After all functions have been calculated the B register is shifted by the serial port to the shift-register/driver. (While some outputs may glitch as data is shifted through them, at 1 Megabaud most people wouldn't notice. Some shift registers provide an "enable" bit to hold the output states while new data is being shifted in.)

This is where the earlier decision to address bits symbolically throughout the program is going to pay off. This major I/O restructuring is nearly as simple to implement as rearranging the input pins. Again, only the bit declarations need to be changed.

```

L_FRNT BIT B.0 : FRONT LEFT-TURN
INDICATOR
R_FRNT BIT B.1 : FRONT RIGHT-TURN
INDICATOR
L_DASH BIT B.2 : DASHBOARD LEFT-TURN
INDICATOR
R_DASH BIT B.3 : DASHBOARD RIGHT-TURN
INDICATOR

```

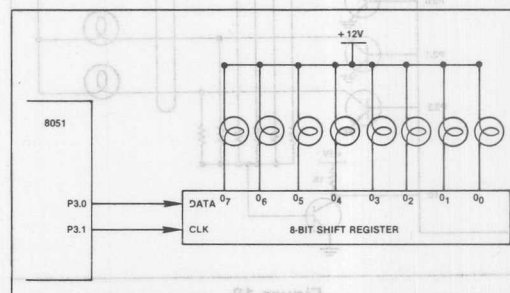


Figure 17. Output expansion using serial port.

```

L_REAR BIT B.4 : REAR LEFT-TURN
INDICATOR
R_REAR BIT B.5 : REAR RIGHT-TURN
INDICATOR

```

The original program to compute the functions need not change. After computing the output variables, the control map is transmitted to the buffered shift register through the serial port:

```
MOV SBUF,B : LOAD BUFFER AND TRANSMIT
```

The Boolean Processor solution holds a number of advantages over older methods. Fewer switches are required. Each is simpler, requiring fewer poles and lower current contacts. The flasher relay is eliminated entirely. Only six filaments are driven, rather than 10. The wiring harness is therefore simpler and less expensive—one conductor for each of the six lamps and each of the five sensor switches. The fewer conductors use far fewer connectors. The whole system is more reliable.

And since the system is much simpler it would be feasible to implement redundancy and/or fault detection on the four main turn indicators. Each could still be a standard double filament bulb, but with the filaments driven in parallel to tolerate single-element failures.

Even with redundancy, the lights will eventually fail. To handle this inescapable fact current or voltage sensing

circuits on each main drive wire can verify that each bulb and its high-current driver is functioning properly. Figure 18 shows one such circuit.

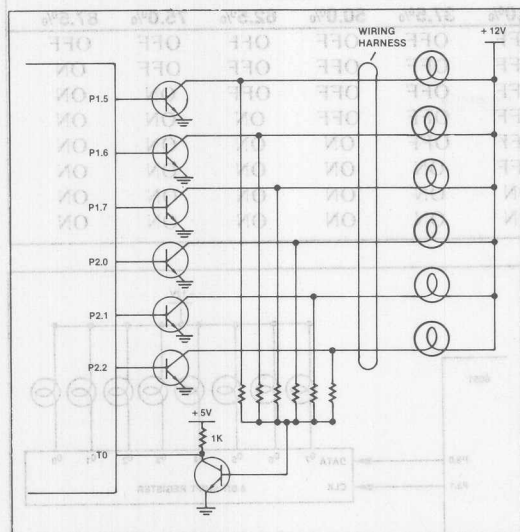


Figure 18.

Assume all of the lights are turned on except one; i.e., all but one of the collectors are grounded. For the bulb which is turned off, if there is continuity from +12 V through the bulb base and filament, the control wire, all connectors, and the P.C. board traces, and if the transistor is indeed not shorted to ground, then the collector will be pulled to +12 V. This turns on the base of Q8 through the corresponding resistor, and grounds the input pin, verifying that the bulb circuit is operational. The continuity of each circuit can be checked by software in this way:

Now turn *all* the bulbs on, grounding all the collectors. Q7 should be turned off, and the Test pin should be high. However, a control wire shorted to +12 V or an open-circuited drive transistor would leave one of the collectors at the higher voltage even now! This too would turn on Q7, indicating a different type of failure. Software could perform these checks once per second by executing the routine every time the software counter SUB_DIV is reloaded by the interrupt routine:

```
DJNZ SUB_DIV,TOSERV
MOV SUB_DIV,#244      ; RELOAD COUNTER
ORL P1,#11100000B     ; SET CONTROL
                        ; OUTPUTS HIGH
ORL P2,#00000111B
CLR L_FRNT             ; FLOAT DRIVE
                        ; COLLECTOR
JB T0,FAULT            ; T0 SHOULD BE
                        ; PULLED LOW
SETB L_FRNT            ; PULL COLLECTOR
                        ; BACK DOWN
```

```
CLR L_DASH
JB T0,FAULT
SETB L_DASH
CLR L_REAR
JB T0,FAULT
SETB L_REAR
CLR R_FRNT
JB T0,FAULT
SETB R_FRNT
CLR R_DASH
JB T0,FAULT
SETB R_DASH
CLR R_REAR
JB T0,FAULT
SETB R_REAR
```

```
WITH ALL COLLECTORS GROUNDED, T0
SHOULD BE HIGH
IF SO, CONTINUE WITH INTERRUPT ROUTINE.
JB T0,TOSERV
FAULT:      ; ELECTRICAL FAILURE
            ; PROCESSING ROUTINE
            ; (LEFT TO READER'S
            ; IMAGINATION)
TOSERV:     ; CONTINUE WITH
            ; INTERRUPT PROCESSING
```

The complete assembled program listing is printed in Appendix A. The resulting code consists of 67 program statements, not counting declarations and comments, which assemble into 150 bytes of object code. Each pass through the service routine requires (coincidentally) 67 usec, plus 32 usec once per second for the electrical test. If executed every 4 msec as suggested this software would typically reduce the throughput of the background program by less than 2%.

Once a microcomputer has been designed into a system, new features suddenly become virtually free. Software could make the emergency blinkers flash alternately or at a rate faster than the turn signals. Turn signals could override the emergency blinkers. Adding more bulbs would allow multiple tail light sequencing and syncopation — true flash factor, so to speak.

Design Example #5 - Complex Control Functions

Finally, we'll mix byte and bit operations to extend the use of 8051 into extremely complex applications.

Programmers can arbitrarily assign I/O pins to input and output functions only if the total does not exceed 32, which is insufficient for applications with a very large number of input variables. One way to expand the number of inputs is with a technique similar to multiplexed-keyboard scanning.

Figure 19 shows a block diagram for a moderately complex programmable industrial controller with the following characteristics:

- 64 input variable sensors;
- 12 output signals;
- Combinational and sequential logic computations;
- Remote operation with communications to a host processor via a high-speed full-duplex serial link;
- Two prioritized external interrupts;
- Internal real-time and time-of-day clocks.

While many microprocessors could be programmed to provide these capabilities with assorted peripheral support chips, an 8051 microcomputer needs **no** other integrated circuits!

The 64 input sensors are logically arranged as an 8x8 matrix. The pins of Port 1 sequentially enable each column of the sensor matrix; as each is enabled Port 0 reads in the state of each sensor in that column. An eight-byte block in bit-addressable RAM remembers the data as it is read in so that after each complete scan cycle there is an internal map of the current state of all sensors. Logic functions can then directly address the elements of the bit map.

The computer's serial port is configured as a nine-bit UART, transferring data at 17,000 bytes-per-second. The ninth bit may distinguish between address and data bytes.

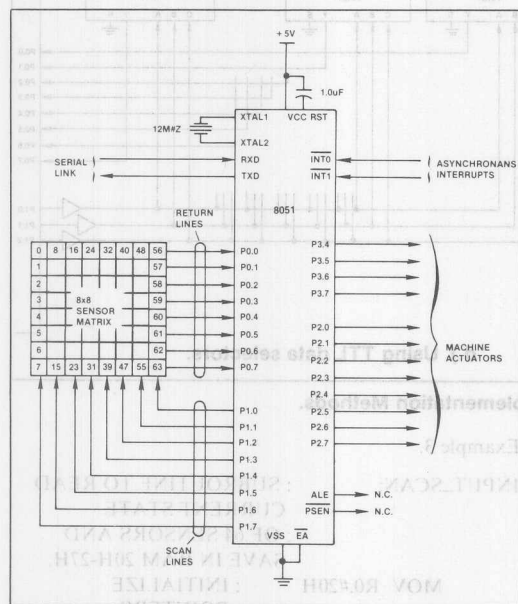


Figure 19. Block diagram of 64-input machine controller.

The 8051 serial port can be configured to detect bytes with the address bit set, automatically ignoring all others. Pins INT0 and INT1 are interrupts configured respectively as high-priority, falling-edge triggered and low-priority, low-level triggered. The remaining 12 I/O pins output TTL-level control signals to 12 actuators.

There are several ways to implement the sensor matrix circuitry, all logically similar. Figure 20.a shows one possibility. Each of the 64 sensors consists of a pair of simple switch contacts in series with a diode to permit multiple contact closures throughout the matrix.

The scan lines from Port 1 provide eight un-encoded active-high scan signals for enabling columns of the matrix. The return lines on rows where a contact is closed are pulled high and read as logic ones. Open return lines are pulled to ground by one of the 40 kohm resistors and are read as zeroes. (The resistor values must be chosen to ensure all return lines are pulled above the 2.0 V logic threshold, even in the worst-case, where all contacts in an enabled column are closed.) Since P0 is provided open-collector outputs and high-impedance MOS inputs its input loading may be considered negligible.

The circuits in Figures 20.b—20.d are variations on this theme. When input signals must be electrically isolated from the computer circuitry as in noisy industrial environments, phototransistors can replace the switch/diode pairs **and** provide optical isolation as in Figure 20.b. Additional opto-isolators could also be used on the control output and special signal lines.

The other circuits assume that input signals are already at TTL levels. Figure 20.c uses octal three-state buffers enabled by active-low scan signals to gate eight signals onto Port 0. Port 0 is available for memory expansion or peripheral chip interfacing between sensor matrix scans. Eight-to-one multiplexers in Figure 20.d select one of eight inputs for each return line as determined by encoded address bits output on three pins of Port 1. (Five more output pins are thus freed for more control functions.) Each output can drive at least one standard TTL or up to 10 low-power TTL loads without additional buffering.

Going back to the original matrix circuit, Figure 21 shows the method used to scan the sensor matrix. Two complete bit maps are maintained in the bit-addressable region of the RAM: one for the current state and one for the previous state read for each sensor. If the need arises, the program could then sense input transitions and/or debounce contact closures by comparing each bit with its earlier value.

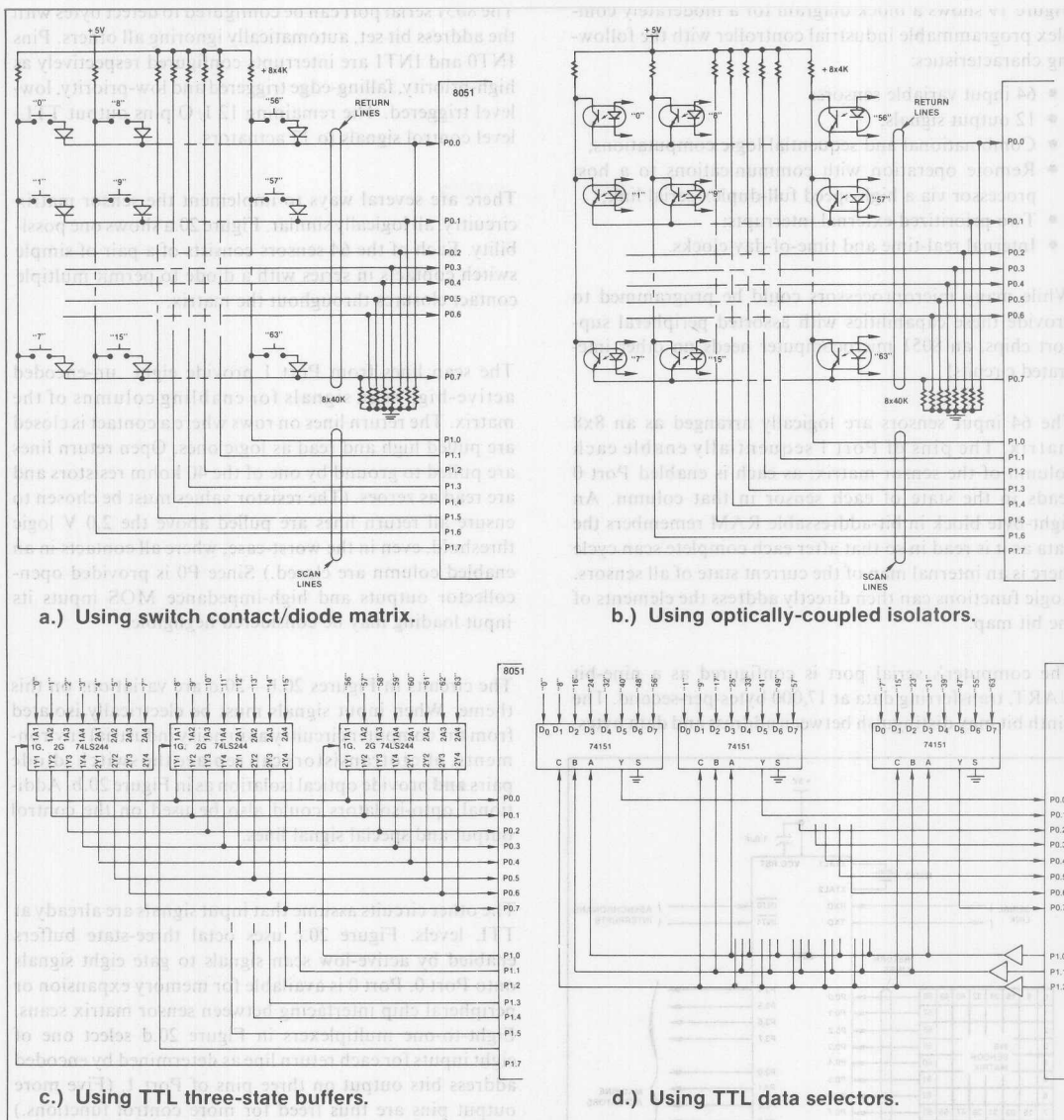


Figure 20. Sensor Matrix Implementation Methods.

Going back to the original matrix circuit, Figure 21 shows the method used to scan the sensor matrix. The complete bit maps are maintained in the bit-addressable region of the code in Example 3 implements the scanning algorithm for the circuits in Figure 20.a. Each column is enabled by setting a single bit in a field of zeroes. The bit maps are positive logic: ones represent contacts that are closed or isolators turned on.

Example 3.

```

INPUT_SCAN:      ; SUBROUTINE TO READ
                  ; CURRENT STATE
                  ; OF 64 SENSORS AND
                  ; SAVE IN RAM 20H-27H.
                  ; INITIALIZE
                  ; POINTERS
MOV R0,#20H      ; FOR BIT MAP
MOV R1,#28H      ; BASES.

```



```

SCAN: MOV P1.A      : OUTPUT TO SCAN
      RR A          : LINES. OR SET
      MOV R2.A      : SHIFT TO ENABLE
      MOV R2.A      : NEXT COLUMN
      MOV R2.A      : NEXT.
      MOV R2.A      : REMEMBER CUR-
      MOV A.P0      : RENT SCAN
      XCH A.@R0     : POSITION.
      MOV @R1.A     : SWITCH WITH
      INC R0        : PREVIOUS MAP
      INC R1        : BITS.
      MOV A.R2      : SAVE PREVIOUS
      JNB ACC.7.SCAN : STATE AS WELL.
      RET          : BUMP POINTERS.

```

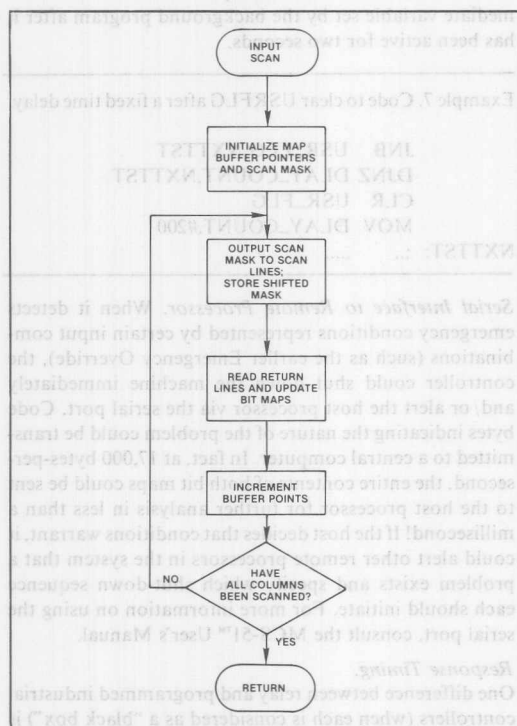


Figure 21. Flowchart for reading in sensor matrix.

inventing some artificial design problem, software corresponding to commonplace logic elements will be discussed.

Combinatorial Output Variables. An output variable which is a simple (or not so simple) combinational function of several input variables is computed in the spirit of Design Example 3. All 64 inputs are represented in the bit maps; in fact, the sensor numbers in Figure 20 correspond to the absolute bit addresses in RAM! The code in Example 4 activates an actuator connected to P2.2 when sensors 12, 23, and 34 are closed and sensors 45 and 56 are open.

Example 4.

Simple Combinatorial Output Variables.

```

: SET P2.2 = (12) (23) (34) ( 45) ( 56)
MOV C.12
ANL C.23
ANL C.34
ANL C. 45
ANL C. 56
MOV P2.2.C

```

Intermediate Variables. The examination of a typical relay-logic ladder diagram will show that many of the rungs control *not* outputs but rather relays whose contacts figure into the computation of other functions. In effect, these relays indicate the state of intermediate variables of a computation.

The MCS-51™ solution can use any directly addressable bit for the storage of such intermediate variables. Even when all 128 bits of the RAM array are dedicated (to input bit maps in this example), the accumulator, PSW, and B register provide 18 additional flags for intermediate variables.

For example, suppose switches 0 through 3 control a safety interlock system. Closing any of them should deactivate certain outputs. Figure 22 is a ladder diagram for this situation. The interlock function could be recomputed for every output affected, or it may be computed once and saved (as implied by the diagram). As the program proceeds this bit can qualify each output.

Example 5. Incorporating Override signal into actuator outputs.

```

CALL INPUT_SCAN
MOV C.0
ORL C.1
ORL C.2
ORL C.3
MOV F0.C

```

```

COMPUTE FUNCTION 0
ANL C,F0
MOV PI.0.C
.....
COMPUTE FUNCTION 1
ANL C,F0
MOV PI.1.C
.....
COMPUTE FUNCTION 2
ANL C,F0
MOV PI.2.C
.....

```

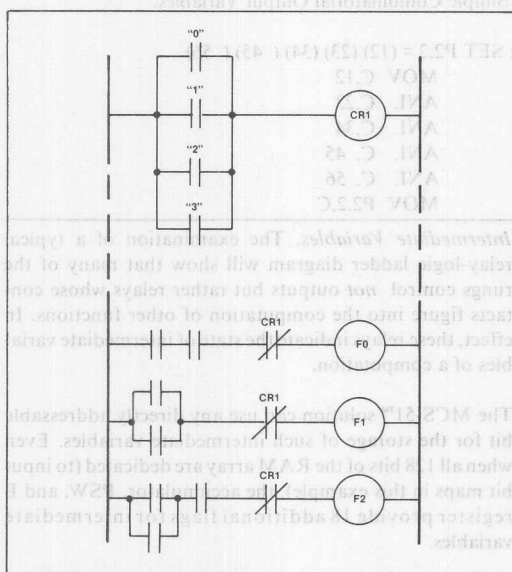


Figure 22. Ladder diagram for output override circuitry.

Latching Relays. A latching relay can be forced into either the ON or OFF state by two corresponding input signals, where it will remain until forced onto the opposite state— analogous to a TTL Set/Reset flip-flop. The relay is used as an intermediate variable for other calculations. In the previous example, the emergency condition could be remembered and remain active until an “emergency cleared” button is pressed.

Any flag or addressable bit may represent a latching relay with a few lines of code (see Example 6).

Example 6. Simulating a latching relay.

```

:LSET SET FLAG 0 IF C=1
LSET ORL C,F0
MOV F0.C
.....
:LRESET RESET FLAG 0 IF C=1
LRESET CPS C
ANL C,F0
MOV F0.C
.....

```

Time Delay Relays. A time delay relay does not respond to an input signal until it has been present (or absent) for some predefined time. For example, a ballast or load resistor may be switched in series with a D.C. motor when it is first turned on, and shunted from the circuit after one second. This sort of time delay may be simulated by an interrupt routine driven by one of the two 8051 timer/counters. The procedure followed by the routine depends heavily on the details of the exact function needed; time-outs or time delays with resettable or non-resettable inputs are possible. If the interrupt routine is executed every 10 milliseconds the code in Example 7 will clear an intermediate variable set by the background program after it has been active for two seconds.

Example 7. Code to clear USRFLG after a fixed time delay.

```

JNB USR_FLG,NXTTST
DJNZ DELAY_COUNT,NXTTST
CLR USR_FLG
MOV DELAY_COUNT,#200
NXTTST: ...

```

Serial Interface to Remote Processor. When it detects emergency conditions represented by certain input combinations (such as the earlier Emergency Override), the controller could shut down the machine immediately and/or alert the host processor via the serial port. Code bytes indicating the nature of the problem could be transmitted to a central computer. In fact, at 17,000 bytes-per-second, the entire contents of both bit maps could be sent to the host processor for further analysis in less than a millisecond! If the host decides that conditions warrant, it could alert other remote processors in the system that a problem exists and specify which shut-down sequence each should initiate. For more information on using the serial port, consult the MCS-51™ User's Manual.

Response Timing.

One difference between relay and programmed industrial controllers (when each is considered as a “black box”) is their respective reaction times to input changes. As reflected by a ladder diagram, relay systems contain a

large number of "rungs" operating in parallel. A change in input conditions will begin propagating through the system immediately, possibly affecting the output state within milliseconds.

Software, on the other hand, operates sequentially. A change in input states will not be detected until the next time an input scan is performed, and will not affect the outputs until that section of the program is reached. For that reason the raw speed of computing the logical functions is of extreme importance.

Here the Boolean processor pays off. *Every instruction mentioned in this Note* completes in one or two microseconds—the *minimum* instruction execution time for many other microcontrollers! A ladder diagram containing a hundred rungs, with an average of four contacts per rung can be replaced by approximately five hundred lines of software. A complete pass through the entire matrix scanning routine and all computations would require about a millisecond; less than the time it takes for most relays to change state.

A programmed controller which simulates each Boolean function with a subroutine would be less efficient by at least an order of magnitude. Extra software is needed for the simulation routines, and each step takes longer to execute for three reasons: several byte-wide logical instructions are executed per user program step (rather than one Boolean operation); most of those instructions take longer to execute with microprocessors performing multiple off-chip accesses; and calling and returning from the various subroutines requires overhead for stack operations.

In fact, the speed of the Boolean Processor solution is likely to be much faster than the system requires. The CPU might use the time left over to compute feedback parameters, collect and analyze execution statistics, perform system diagnostics, and so forth.

Additional functions and uses.

With the building-block basics mentioned above many more operations may be synthesized by short instruction sequences.

Exclusive-OR. There are no common mechanical devices or relays analogous to the Exclusive-OR operation, so this instruction was omitted from the Boolean Processor. However, the Exclusive-OR or Exclusive-NOR operation may be performed in two instructions by conditionally complementing the carry or a Boolean variable based on the state of any other testable bit.

EXCLUSIVE-OR FUNCTION IMPOSED ON CARRY USING F0 IS INPUT VARIABLE.

```
XOR_F0: JNB F0,XORCNT ; ("JB" FOR X-NOR)
        CPL C
XORCNT:...
```

XCH. The contents of the carry and some other bit may be exchanged (switched) by using the accumulator as temporary storage. Bits can be moved into and out of the accumulator simultaneously using the Rotate-through-carry instructions, though this would alter the accumulator data.

EXCHANGE CARRY WITH USRFLG

```
XCHBIT: RLC A
        MOV C,USR_FLG
        RRC A
        MOV USR_FLG,C
        RLC A
```

Extended Bit Addressing. The 8051 can directly address 144 general-purpose bits for all instructions in Figure 3.b. Similar operations may be extended to any bit anywhere on the chip with some loss of efficiency.

The logical operations AND, OR, and Exclusive-OR are performed on byte variables using six different addressing modes, one of which lets the source be an immediate mask, and the destination any directly addressable byte. Any bit may thus be set, cleared, or complemented with a three-byte, two-cycle instruction if the mask has all bits but one set or cleared.

Byte variables, registers, and indirectly addressed RAM may be moved to a bit addressable register (usually the accumulator) in one instruction. Once transferred, the bits may be tested with a conditional jump, allowing any bit to be polled in 3 microseconds—still much faster than most architectures—or used for logical calculations. (This technique can also simulate additional bit addressing modes with byte operations.)

Parity of bytes or bits. The parity of the current accumulator contents is always available in the PSW, from whence it may be moved to the carry and further processed. Error-correcting Hamming codes and similar applications require computing parity on groups of isolated bits. This can be done by conditionally complementing the carry flag based on those bits or by gathering the bits into the accumulator (as shown in the DES example) and then testing the parallel parity flag.

Multiple byte shift and CRC codes.

Though the 8051 serial port can accommodate eight- or nine-bit data transmissions, some protocols involve much

longer bit streams. The algorithms presented in Design Example 2 can be extended quite readily to 16 or more bits by using multi-byte input and output buffers.

Many mass data storage peripherals and serial communications protocols include Cyclic Redundancy (CRC) codes to verify data integrity. The function is generally computed serially by hardware using shift registers and Exclusive-OR gates, but it can be done with software. As each bit is received into the carry, appropriate bits in the multi-byte data buffer are conditionally complemented based on the incoming data bit. When finished, the CRC register contents may be checked for zero by ORing the two bytes in the accumulator.

4. SUMMARY

A truly unique facet of the Intel MCS-51™ microcomputer family design is the collection of features optimized for the one-bit operations so often desired in real-world, real-time control applications. Included are 17 special instructions, a Boolean accumulator, implicit and direct addressing modes, program and mass data storage, and many I/O options. These are the world's first single-chip microcomputers able to efficiently manipulate, operate on, and transfer either bytes or individual bits as data.

This Application Note has detailed the information needed by a microcomputer system designer to make full use of these capabilities. Five design examples were used to contrast the solutions allowed by the 8051 and those required by previous architectures. Depending on the individual application, the 8051 solution will be easier to design, more reliable to implement, debug, and verify, use less program memory, and run up to an order of magnitude faster than the same function implemented on previous digital computer architectures.

Combining byte- and bit-handling capabilities in a single microcomputer has a strong synergistic effect: the power of the result exceeds the power of byte- and bit-processors laboring individually. Virtually all user applications will benefit in some ways from this duality. Data intensive applications will use bit addressing for test pin monitoring or program control flags; control applications will use byte manipulation for parallel I/O expansion or arithmetic calculations.

It is hoped that these design examples give the reader an appreciation of these unique features and suggest ways to exploit them in his or her own application.

The logical operations AND, OR, and Exclusive-OR are performed on byte variables using six different addressing modes, one of which lets the source be an immediate mask, and the destination any directly addressable byte. Any bit may thus be set, cleared, or complemented with a three-byte, two-cycle instruction if the mask has all bits but one set or cleared.

Byte variables, registers, and indirectly addressed RAM may be moved to a bit addressable register (usually the accumulator) in one instruction. Once transferred, the bits may be tested with a conditional jump allowing any bit to be polled in 3 microseconds—still much faster than most architectures—or used for logical calculations. (This technique can also simulate additional bit addressing modes with byte operations.)

Four of bytes or bits. The parity of the current accumulator contents is always available in the P2W, from whence it may be moved to the carry and further processed. Error-correcting Hamming codes and similar applications require computing parity on groups of isolated bits. This can be done by conditionally complementing the carry flag based on those bits or by gathering the bits into the accumulator (as shown in the DES example) and then testing the parallel parity flag.

Multiple byte shift and CRC codes. Though the 8051 serial port can accommodate eight- or nine-bit data transmissions, some protocols involve much

In fact, the speed of the Boolean Processor solution is likely to be much faster than the system requires. The CPU might use the time left over to compute feedback parameters, collect and analyze execution statistics, perform system diagnostics, and so forth.

Additional functions and uses.

With the building-block basics mentioned above many more operations may be synthesized by short instruction sequences.

Exclusive-OR. There are no common mechanical devices or relays analogous to the Exclusive-OR operation, so this instruction was omitted from the Boolean Processor. However, the Exclusive-OR or Exclusive-NOR operation may be performed in two instructions by conditionally complementing the carry or a Boolean variable based on the state of any other testable bit.

ISIS-II MCS-51 MACRO ASSEMBLER V1.0
OBJECT MODULE PLACED IN :FO:AP70.HEX
ASSEMBLER INVOKED BY : f1:asm51 ap70.src date(328)

```

LOC OBJ      LINE      SOURCE
2-73
0000 0000      1      $XREF TITLE(AP-70 APPENDIX)
0001 0000      2      ; *****
0002 0000      3      ;
0003 0000      4      ;
0004 0000      5      ; THE FOLLOWING PROGRAM USES THE BOOLEAN INSTRUCTION SET
0005 0000      6      ; OF THE INTEL 8051 MICROCOMPUTER TO PERFORM A NUMBER OF
0006 0000      7      ; AUTOMOTIVE DASHBOARD CONTROL FUNCTIONS RELATING TO
0007 0000      8      ; TURN SIGNAL CONTROL, EMERGENCY BLINKERS, BRAKE LIGHT
0008 0000      9      ; CONTROL, AND PARKING LIGHT OPERATION.
0009 0000     10      ; THE ALGORITHMS AND HARDWARE ARE DESCRIBED IN DESIGN
0010 0000     11      ; EXAMPLE #4 OF INTEL APPLICATION NOTE AP-70,
0011 0000     12      ; "USING THE INTEL MCS-51(TM)
0012 0000     13      ; BOOLEAN PROCESSING CAPABILITIES"
0013 0000     14      ; *****
0014 0000     15      ;
0015 0000     16      ; INPUT PIN DECLARATIONS:
0016 0000     17      ; (ALL INPUTS ARE POSITIVE-TRUE LOGIC.
0017 0000     18      ; INPUTS ARE HIGH WHEN RESPECTIVE SWITCH CONTACT IS CLOSED.)
0018 0000     19      ;
0019 0000     20      BRAKE BIT P1.0 ; BRAKE PEDAL DEPRESSED
0020 0000     21      EMERG BIT P1.1 ; EMERGENCY BLINKER ACTIVATED
0021 0000     22      PARK BIT P1.2 ; PARKING LIGHTS ON
0022 0000     23      L_TURN BIT P1.3 ; TURN LEVER DOWN
0023 0000     24      R_TURN BIT P1.4 ; TURN LEVER UP
0024 0000     25      ;
0025 0000     26      ; OUTPUT PIN DECLARATIONS:
0026 0000     27      ; (ALL OUTPUTS ARE POSITIVE TRUE LOGIC.
0027 0000     28      ; BULB IS TURNED ON WHEN OUTPUT PIN IS HIGH.)
0028 0000     29      ;
0029 0000     30      L_FRNT BIT P1.5 ; FRONT LEFT-TURN INDICATOR
0030 0000     31      R_FRNT BIT P1.6 ; FRONT RIGHT-TURN INDICATOR
0031 0000     32      L_DASH BIT P1.7 ; DASHBOARD LEFT-TURN INDICATOR
0032 0000     33      R_DASH BIT P2.0 ; DASHBOARD RIGHT-TURN INDICATOR
0033 0000     34      L_REAR BIT P2.1 ; REAR LEFT-TURN INDICATOR
0034 0000     35      R_REAR BIT P2.2 ; REAR RIGHT-TURN INDICATOR
0035 0000     36      ;
0036 0000     37      SYS_FAIL BIT P2.3 ; ELECTRICAL SYSTEM FAULT INDICATOR
0037 0000     38      ;
0038 0000     39      ; INTERNAL VARIABLE DEFINITIONS:
0039 0000     40      ;
0040 0000     41      SUB_DIV DATA 20H ; INTERRUPT RATE SUBDIVIDER
0041 0000     42      HI_FREQ BIT SUB_DIV.0 ; HIGH-FREQUENCY OSCILLATOR BIT
0042 0000     43      LO_FREQ BIT SUB_DIV.7 ; LOW-FREQUENCY OSCILLATOR BIT
0043 0000     44      ;
0044 0000     45      DIM BIT P0SW.1 ; PARKING LIGHTS ON FLAG
0045 0000     46      ;
0046 0000     47      ; *****
0047 0000     48      $EJECT
0048 0000
0049 0000
0050 0000
0051 0000
0052 0000
0053 0000
0054 0000
0055 0000
0056 0000
0057 0000
0058 0000
0059 0000
0060 0000
0061 0000
0062 0000
0063 0000
0064 0000
0065 0000
0066 0000
0067 0000
0068 0000
0069 0000
0070 0000
0071 0000
0072 0000
0073 0000
0074 0000
0075 0000
0076 0000
0077 0000
0078 0000
0079 0000
0080 0000
0081 0000
0082 0000
0083 0000
0084 0000
0085 0000
0086 0000
0087 0000
0088 0000
0089 0000
0090 0000
0091 0000
0092 0000
0093 0000
0094 0000
0095 0000
0096 0000
0097 0000
0098 0000
0099 0000
0100 0000
0101 0000
0102 0000
0103 0000
0104 0000
0105 0000
0106 0000
0107 0000
0108 0000
0109 0000
0110 0000
0111 0000
0112 0000
0113 0000
0114 0000
0115 0000
0116 0000
0117 0000
0118 0000
0119 0000
0120 0000
0121 0000
0122 0000
0123 0000
0124 0000
0125 0000
0126 0000
0127 0000
0128 0000
0129 0000
0130 0000
0131 0000
0132 0000
0133 0000
0134 0000
0135 0000
0136 0000
0137 0000
0138 0000
0139 0000
0140 0000
0141 0000
0142 0000
0143 0000
0144 0000
0145 0000
0146 0000
0147 0000
0148 0000
0149 0000
0150 0000
0151 0000
0152 0000
0153 0000
0154 0000
0155 0000
0156 0000
0157 0000
0158 0000
0159 0000
0160 0000
0161 0000
0162 0000
0163 0000
0164 0000
0165 0000
0166 0000
0167 0000
0168 0000
0169 0000
0170 0000
0171 0000
0172 0000
0173 0000
0174 0000
0175 0000
0176 0000
0177 0000
0178 0000
0179 0000
0180 0000
0181 0000
0182 0000
0183 0000
0184 0000
0185 0000
0186 0000
0187 0000
0188 0000
0189 0000
0190 0000
0191 0000
0192 0000
0193 0000
0194 0000
0195 0000
0196 0000
0197 0000
0198 0000
0199 0000
0200 0000
0201 0000
0202 0000
0203 0000
0204 0000
0205 0000
0206 0000
0207 0000
0208 0000
0209 0000
0210 0000
0211 0000
0212 0000
0213 0000
0214 0000
0215 0000
0216 0000
0217 0000
0218 0000
0219 0000
0220 0000
0221 0000
0222 0000
0223 0000
0224 0000
0225 0000
0226 0000
0227 0000
0228 0000
0229 0000
0230 0000
0231 0000
0232 0000
0233 0000
0234 0000
0235 0000
0236 0000
0237 0000
0238 0000
0239 0000
0240 0000
0241 0000
0242 0000
0243 0000
0244 0000
0245 0000
0246 0000
0247 0000
0248 0000
0249 0000
0250 0000
0251 0000
0252 0000
0253 0000
0254 0000
0255 0000
0256 0000
0257 0000
0258 0000
0259 0000
0260 0000
0261 0000
0262 0000
0263 0000
0264 0000
0265 0000
0266 0000
0267 0000
0268 0000
0269 0000
0270 0000
0271 0000
0272 0000
0273 0000
0274 0000
0275 0000
0276 0000
0277 0000
0278 0000
0279 0000
0280 0000
0281 0000
0282 0000
0283 0000
0284 0000
0285 0000
0286 0000
0287 0000
0288 0000
0289 0000
0290 0000
0291 0000
0292 0000
0293 0000
0294 0000
0295 0000
0296 0000
0297 0000
0298 0000
0299 0000
0300 0000
0301 0000
0302 0000
0303 0000
0304 0000
0305 0000
0306 0000
0307 0000
0308 0000
0309 0000
0310 0000
0311 0000
0312 0000
0313 0000
0314 0000
0315 0000
0316 0000
0317 0000
0318 0000
0319 0000
0320 0000
0321 0000
0322 0000
0323 0000
0324 0000
0325 0000
0326 0000
0327 0000
0328 0000
0329 0000
0330 0000
0331 0000
0332 0000
0333 0000
0334 0000
0335 0000
0336 0000
0337 0000
0338 0000
0339 0000
0340 0000
0341 0000
0342 0000
0343 0000
0344 0000
0345 0000
0346 0000
0347 0000
0348 0000
0349 0000
0350 0000
0351 0000
0352 0000
0353 0000
0354 0000
0355 0000
0356 0000
0357 0000
0358 0000
0359 0000
0360 0000
0361 0000
0362 0000
0363 0000
0364 0000
0365 0000
0366 0000
0367 0000
0368 0000
0369 0000
0370 0000
0371 0000
0372 0000
0373 0000
0374 0000
0375 0000
0376 0000
0377 0000
0378 0000
0379 0000
0380 0000
0381 0000
0382 0000
0383 0000
0384 0000
0385 0000
0386 0000
0387 0000
0388 0000
0389 0000
0390 0000
0391 0000
0392 0000
0393 0000
0394 0000
0395 0000
0396 0000
0397 0000
0398 0000
0399 0000
0400 0000
0401 0000
0402 0000
0403 0000
0404 0000
0405 0000
0406 0000
0407 0000
0408 0000
0409 0000
0410 0000
0411 0000
0412 0000
0413 0000
0414 0000
0415 0000
0416 0000
0417 0000
0418 0000
0419 0000
0420 0000
0421 0000
0422 0000
0423 0000
0424 0000
0425 0000
0426 0000
0427 0000
0428 0000
0429 0000
0430 0000
0431 0000
0432 0000
0433 0000
0434 0000
0435 0000
0436 0000
0437 0000
0438 0000
0439 0000
0440 0000
0441 0000
0442 0000
0443 0000
0444 0000
0445 0000
0446 0000
0447 0000
0448 0000
0449 0000
0450 0000
0451 0000
0452 0000
0453 0000
0454 0000
0455 0000
0456 0000
0457 0000
0458 0000
0459 0000
0460 0000
0461 0000
0462 0000
0463 0000
0464 0000
0465 0000
0466 0000
0467 0000
0468 0000
0469 0000
0470 0000
0471 0000
0472 0000
0473 0000
0474 0000
0475 0000
0476 0000
0477 0000
0478 0000
0479 0000
0480 0000
0481 0000
0482 0000
0483 0000
0484 0000
0485 0000
0486 0000
0487 0000
0488 0000
0489 0000
0490 0000
0491 0000
0492 0000
0493 0000
0494 0000
0495 0000
0496 0000
0497 0000
0498 0000
0499 0000
0500 0000
0501 0000
0502 0000
0503 0000
0504 0000
0505 0000
0506 0000
0507 0000
0508 0000
0509 0000
0510 0000
0511 0000
0512 0000
0513 0000
0514 0000
0515 0000
0516 0000
0517 0000
0518 0000
0519 0000
0520 0000
0521 0000
0522 0000
0523 0000
0524 0000
0525 0000
0526 0000
0527 0000
0528 0000
0529 0000
0530 0000
0531 0000
0532 0000
0533 0000
0534 0000
0535 0000
0536 0000
0537 0000
0538 0000
0539 0000
0540 0000
0541 0000
0542 0000
0543 0000
0544 0000
0545 0000
0546 0000
0547 0000
0548 0000
0549 0000
0550 0000
0551 0000
0552 0000
0553 0000
0554 0000
0555 0000
0556 0000
0557 0000
0558 0000
0559 0000
0560 0000
0561 0000
0562 0000
0563 0000
0564 0000
0565 0000
0566 0000
0567 0000
0568 0000
0569 0000
0570 0000
0571 0000
0572 0000
0573 0000
0574 0000
0575 0000
0576 0000
0577 0000
0578 0000
0579 0000
0580 0000
0581 0000
0582 0000
0583 0000
0584 0000
0585 0000
0586 0000
0587 0000
0588 0000
0589 0000
0590 0000
0591 0000
0592 0000
0593 0000
0594 0000
0595 0000
0596 0000
0597 0000
0598 0000
0599 0000
0600 0000
0601 0000
0602 0000
0603 0000
0604 0000
0605 0000
0606 0000
0607 0000
0608 0000
0609 0000
0610 0000
0611 0000
0612 0000
0613 0000
0614 0000
0615 0000
0616 0000
0617 0000
0618 0000
0619 0000
0620 0000
0621 0000
0622 0000
0623 0000
0624 0000
0625 0000
0626 0000
0627 0000
0628 0000
0629 0000
0630 0000
0631 0000
0632 0000
0633 0000
0634 0000
0635 0000
0636 0000
0637 0000
0638 0000
0639 0000
0640 0000
0641 0000
0642 0000
0643 0000
0644 0000
0645 0000
0646 0000
0647 0000
0648 0000
0649 0000
0650 0000
0651 0000
0652 0000
0653 0000
0654 0000
0655 0000
0656 0000
0657 0000
0658 0000
0659 0000
0660 0000
0661 0000
0662 0000
0663 0000
0664 0000
0665 0000
0666 0000
0667 0000
0668 0000
0669 0000
0670 0000
0671 0000
0672 0000
0673 0000
0674 0000
0675 0000
0676 0000
0677 0000
0678 0000
0679 0000
0680 0000
0681 0000
0682 0000
0683 0000
0684 0000
0685 0000
0686 0000
0687 0000
0688 0000
0689 0000
0690 0000
0691 0000
0692 0000
0693 0000
0694 0000
0695 0000
0696 0000
0697 0000
0698 0000
0699 0000
0700 0000
0701 0000
0702 0000
0703 0000
0704 0000
0705 0000
0706 0000
0707 0000
0708 0000
0709 0000
0710 0000
0711 0000
0712 0000
0713 0000
0714 0000
0715 0000
0716 0000
0717 0000
0718 0000
0719 0000
0720 0000
0721 0000
0722 0000
0723 0000
0724 0000
0725 0000
0726 0000
0727 0000
0728 0000
0729 0000
0730 0000
0731 0000
0732 0000
0733 0000
0734 0000
0735 0000
0736 0000
0737 0000
0738 0000
0739 0000
0740 0000
0741 0000
0742 0000
0743 0000
0744 0000
0745 0000
0746 0000
0747 0000
0748 0000
0749 0000
0750 0000
0751 0000
0752 0000
0753 0000
0754 0000
0755 0000
0756 0000
0757 0000
0758 0000
0759 0000
0760 0000
0761 0000
0762 0000
0763 0000
0764 0000
0765 0000
0766 0000
0767 0000
0768 0000
0769 0000
0770 0000
0771 0000
0772 0000
0773 0000
0774 0000
0775 0000
0776 0000
0777 0000
0778 0000
0779 0000
0780 0000
0781 0000
0782 0000
0783 0000
0784 0000
0785 0000
0786 0000
0787 0000
0788 0000
0789 0000
0790 0000
0791 0000
0792 0000
0793 0000
0794 0000
0795 0000
0796 0000
0797 0000
0798 0000
0799 0000
0800 0000
0801 0000
0802 0000
0803 0000
0804 0000
0805 0000
0806 0000
0807 0000
0808 0000
0809 0000
0810 0000
0811 0000
0812 0000
0813 0000
0814 0000
0815 0000
0816 0000
0817 0000
0818 0000
0819 0000
0820 0000
0821 0000
0822 0000
0823 0000
0824 0000
0825 0000
0826 0000
0827 0000
0828 0000
0829 0000
0830 0000
0831 0000
0832 0000
0833 0000
0834 0000
0835 0000
0836 0000
0837 0000
0838 0000
0839 0000
0840 0000
0841 0000
0842 0000
0843 0000
0844 0000
0845 0000
0846 0000
0847 0000
0848 0000
0849 0000
0850 0000
0851 0000
0852 0000
0853 0000
0854 0000
0855 0000
0856 0000
0857 0000
0858 0000
0859 0000
0860 0000
0861 0000
0862 0000
0863 0000
0864 0000
0865 0000
0866 0000
0867 0000
0868 0000
0869 0000
0870 0000
0871 0000
0872 0000
0873 0000
0874 0000
0875 0000
0876 0000
0877 0000
0878 0000
0879 0000
0880 0000
0881 0000
0882 0000
0883 0000
0884 0000
0885 0000
0886 0000
0887 0000
0888 0000
0889 0000
0890 0000
0891 0000
0892 0000
0893 0000
0894 0000
0895 0000
0896 0000
0897 0000
0898 0000
0899 0000
0900 0000
0901 0000
0902 0000
0903 0000
0904 0000
0905 0000
0906 0000
0907 0000
0908 0000
0909 0000
0910 0000
0911 0000
0912 0000
0913 0000
0914 0000
0915 0000
0916 0000
0917 0000
0918 0000
0919 0000
0920 0000
0921 0000
0922 0000
0923 0000
0924 0000
0925 0000
0926 0000
0927 0000
0928 0000
0929 0000
0930 0000
0931 0000
0932 0000
0933 0000
0934 0000
0935 0000
0936 0000
0937 0000
0938 0000
0939 0000
0940 0000
0941 0000
0942 0000
0943 0000
0944 0000
0945 0000
0946 0000
0947 0000
0948 0000
0949 0000
0950 0000
0951 0000
0952 0000
0953 0000
0954 0000
0955 0000
0956 0000
0957 0000
0958 0000
0959 0000
0960 0000
0961 0000
0962 0000
0963 0000
0964 0000
0965 0000
0966 0000
0967 0000
0968 0000
0969 0000
0970 0000
0971 0000
0972 0000
0973 0000
0974 0000
0975 0000
0976 0000
0977 0000
0978 0000
0979 0000
0980 0000
0981 0000
0982 0000
0983 0000
0984 0000
0985 0000
0986 0000
0987 0000
0988 0000
0989 0000
0990 0000
0991 0000
0992 0000
0993 0000
0994 0000
0995 0000
0996 0000
0997 0000
0998 0000
0999 0000
1000 0000
1001 0000
1002 0000
1003 0000
1004 0000
1005 0000
1006 0000
1007 0000
1008 0000
1009 0000
1010 0000
1011 0000
1012 0000
1013 0000
1014 0000
1015 0000
1016 0000
1017 0000
1018 0000
1019 0000
1020 0000
1021 0000
1022 0000
1023 0000
1024 0000
1025 0000
1026 0000
1027 0000
1028 0000
1029 0000
1030 0000
1031 0000
1032 0000
1033 0000
1034 0000
1035 0000
1036 0000
1037 0000
1038 0000
1039 0000
1040 0000
1041 0000
1042 0000
1043 0000
1044 0000
1045 0000
1046 0000
1047 0000
1048 0000
1049 0000
1050 0000
1051 0000
1052 0000
1053 0000
1054 0000
1055 0000
1056 0000
1057 0000
1058 0000
1059 0000
1060 0000
1061 0000
1062 0000
1063 0000
1064 0000
1065 0000
1066 0000
1067 0000
1068 0000
1069 0000
1070 0000
1071 0000
1072 0000
1073 0000
1074 0000
1075 0000
1076 0000
1077 0000
1078 0000
1079 0000
1080 0000
1081 0000
1082 0000
1083 0000
1084 0000
1085 0000
1086 0000
1087 0000
1088 0000
1089 0000
1090 0000
1091 0000
1092 0000
1093 0000
1094 0000
1095 0000
1096 0000
1097 0000
1098 0000
1099 0000
1100 0000
1101 0000
1102 0000
1103 0000
1104 0000
1105 0000
1106 0000
1107 0000
1108 0000
1109 0000
1110 0000
1111 0000
1112 0000
1113 0000
1114 0000
1115 0000
1116 0000
1117 0000
1118 0000
1119 0000
1120 0000
1121 0000
1122 0000
1123 0000
1124 0000
1125 0000
1126 0000
1127 0000
1128 0000
1129 0000
1130 0000
1131 0000
1132 0000
1133 0000
1134 0000
1135 0000
1136 0000
1137 0000
1138 0000
1139 0000
1140 0000
1141 0000
1142 0000
1143 0000
1144 0000
1145 0000
1146 0000
1147 0000
1148 0000
1149 0000
11
```

LOC OBJ

LINE:8 47 SOURCE:1

```

49      ORG      0000H      ; RESET VECTOR
50      DIM      LUMP1      INIT:1
51      ;
52      ORG      000BH      ; TIMER 0 SERVICE VECTOR
53      MOV      TH0, #-16   ; HIGH TIMER BYTE ADJUSTED TO CONTROL INT. RATE
54      PUSH    PSW          ; EXECUTE CODE TO SAVE ANY REGISTERS USED BELOW
55      AJMP     UPDATE      ; (CONTINUE WITH REST OF ROUTINE)
56      ;
57      ORG      0040H
58      INIT: MOV  TL0, #0    ; ZERO LOADED INTO LOW-ORDER BYTE AND
59      MOV      TH0, #-16   ; -16 IN HIGH-ORDER BYTE GIVES 4 MSEC PERIOD
60      MOV      TMOD, #01100001B ; 8-BIT AUTO RELOAD COUNTER MODE FOR TIMER 1,
61      ; 16-BIT TIMER MODE FOR TIMER 0 SELECTED
62      MOV      SUB_DIV, #244 ; SUBDIVIDE INTERRUPT RATE BY 244 FOR 1 HZ
63      SETB     ET0         ; USE TIMER 0 OVERFLOWS TO INTERRUPT PROGRAM
64      SETB     EA          ; CONFIGURE IE TO GLOBALLY ENABLE INTERRUPTS
65      SETB     TR0         ; KEEP INSTRUCTION CYCLE COUNT UNTIL OVERFLOW
66      SJMP     $          ; START BACKGROUND PROGRAM EXECUTION
67      ;
68      ;
69      UPDATE: DJNZ  SUB_DIV, TOSERV ; EXECUTE SYSTEM TEST ONLY ONCE PER SECOND
70      MOV      SUB_DIV, #244      ; GET VALUE FOR NEXT ONE SECOND DELAY AND
71      ; GO THROUGH ELECTRICAL SYSTEM TEST CODE:
72      ORL      P1, #11100000B    ; SET CONTROL OUTPUTS HIGH
73      ORL      P2, #00000111B    ;
74      CLR      L_FRNT           ; FLOAT DRIVE COLLECTOR
75      JB       TO_FAULT         ; TO SHOULD BE PULLED LOW
76      SETB     L_FRNT          ; PULL COLLECTOR BACK DOWN
77      CLR      L_DASH          ; REPEAT SEQUENCE FOR L_DASH,
78      JB       TO_FAULT
79      SETB     L_DASH
80      CLR      L_REAR          ; L_REAR,
81      JB       TO_FAULT
82      SETB     L_REAR
83      CLR      R_FRNT          ; R_FRNT,
84      JB       TO_FAULT
85      SETB     R_FRNT
86      CLR      R_DASH          ; R_DASH,
87      JB       TO_FAULT
88      SETB     R_DASH
89      CLR      R_REAR          ; AND R_REAR,
90      JB       TO_FAULT
91      SETB     R_REAR
92      ;
93      ; ***** WITH ALL COLLECTORS GROUNDED, TO SHOULD BE HIGH *****
94      ; ***** IF SO, CONTINUE WITH INTERRUPT ROUTINE. *****
95      ;
96      JB       TO_TOSERV
97      FAULT: CPL      S_FAIL    ; ELECTRICAL FAILURE PROCESSING ROUTINE
98      ; (TOGGLE INDICATOR ONCE PER SECOND)
99      +1 $EJECT

```

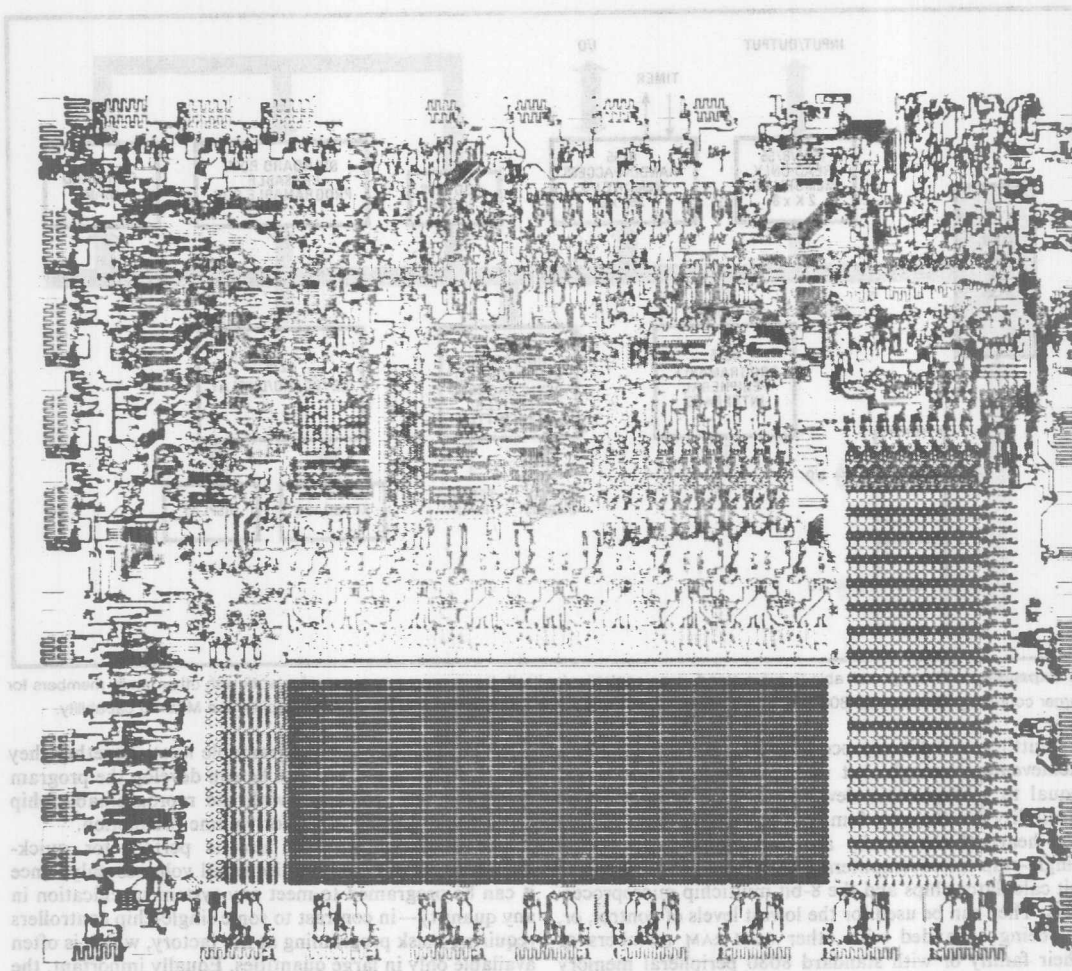
LOC	OBJ	LINE	SOURCE
		100	; CONTINUE WITH INTERRUPT PROCESSING:
		101	;
		102	; 1) COMPUTE LOW BULB INTENSITY WHEN PARKING LIGHTS ARE ON.
		103	;
008F	A201	104	TOSERV: MOV C, SUB_DIV. 1 ; START WITH 50 PERCENT,
0091	8200	105	ANL C, SUB_DIV. 0 ; MASK DOWN TO 25 PERCENT,
0093	7202	106	ORL C, SUB_DIV. 2 ; BUILD BACK TO 62.5 PERCENT,
0095	8292	107	ANL C, PARK ; GATE WITH PARKING LIGHT SWITCH,
0097	92D1	108	MOV DIM, C ; AND SAVE IN TEMP. VARIABLE.
		109	;
		110	; 2) COMPUTE AND OUTPUT LEFT-HAND DASHBOARD INDICATOR.
		111	;
0099	A293	112	MOV C, L_TURN ; SET CARRY IF TURN
009B	7291	113	ORL C, EMERG ; OR EMERGENCY SELECTED.
009D	8207	114	ANL C, LO_FREQ ; IF SO, GATE IN 1 HZ SIGNAL
009F	9297	115	MOV L_DASH, C ; AND OUTPUT TO DASHBOARD.
		116	;
		117	; 3) COMPUTE AND OUTPUT LEFT-HAND FRONT TURN SIGNAL.
		118	;
00A1	92D5	119	MOV FO, C ; SAVE FUNCTION SO FAR.
00A3	72D1	120	ORL C, DIM ; ADD IN PARKING LIGHT FUNCTION
00A5	9295	121	MOV L_FRNT, C ; AND OUTPUT TO TURN SIGNAL.
		122	;
		123	; 4) COMPUTE AND OUTPUT LEFT-HAND REAR TURN SIGNAL.
		124	;
00A7	A290	125	MOV C, BRAKE ; GATE BRAKE PEDAL SWITCH
00A9	8093	126	ANL C, /L_TURN ; WITH TURN LEVER.
00AB	72D5	127	ORL C, FO ; INCLUDE TEMP. VARIABLE FROM DASH
00AD	72D1	128	ORL C, DIM ; AND PARKING LIGHT FUNCTION
00AF	92A1	129	MOV L_REAR, C ; AND OUTPUT TO TURN SIGNAL.
		130	;
		131	; 5) REPEAT ALL OF ABOVE FOR RIGHT-HAND COUNTERPARTS.
		132	;
00B1	A294	133	MOV C, R_TURN ; SET CARRY IF TURN
00B3	7291	134	ORL C, EMERG ; OR EMERGENCY SELECTED.
00B5	8207	135	ANL C, LO_FREQ ; IF SO, GATE IN 1 HZ SIGNAL
00B7	92A0	136	MOV R_DASH, C ; AND OUTPUT TO DASHBOARD.
00B9	92D5	137	MOV FO, C ; SAVE FUNCTION SO FAR.
00BB	72D1	138	ORL C, DIM ; ADD IN PARKING LIGHT FUNCTION
00BD	9296	139	MOV R_FRNT, C ; AND OUTPUT TO TURN SIGNAL.
00BF	A290	140	MOV C, BRAKE ; GATE BRAKE PEDAL SWITCH
00C1	8094	141	ANL C, /R_TURN ; WITH TURN LEVER.
00C3	72D5	142	ORL C, FO ; INCLUDE TEMP. VARIABLE FROM DASH
00C5	72D1	143	ORL C, DIM ; AND PARKING LIGHT FUNCTION
00C7	92A2	144	MOV R_REAR, C ; AND OUTPUT TO TURN SIGNAL.
		145	;
		146	RESTORE STATUS REGISTER AND RETURN.
		147	;
00C9	D0D0	148	POP PSW ; RESTORE PSW
00CB	32	149	RETI ; AND RETURN FROM INTERRUPT ROUTINE
		150	;
		151	END

```

101      END
102
103      00CB 35      104      REEL      1      VMD RELOIN FROM INTERMEDI MOOTIME
104      00CA 0000    105      60#      60#      RESUME 60#
105
106      RESUME SAVING REGISTER AND RELOIN
107
108      00C1 45V5      XREF SYMBOL TABLE LISTING
109      00C2 45D1      110      0000H 20# 125 140      1      VMD ONLY TO LOAN SIGNIF
110      00C3 45D2      111      00D1H 45# 108 120 128 138 143      1      VMD SAVING FIGHT FUNCTION
111      00C4 45D3      112      00AFH 64# 125 140      1      INCLUDE LEAF ACTIVAGE FROM DASH
112      00C5 45D4      113      0091H 21# 113 134      1      WITH LOAN FEEL
113      00C6 45D5      114      00A9H 63# 125 140      1      SAVE SWAKE BEDW SMICH
114      00C7 45D6      115      00D5H 119 127 137 142      1      VMD ONLY TO LOAN SIGNIF
115      00C8 45D7      116      008DH 75 78 81 84 87 90 97#      1      ADD IN SAVING FIGHT FUNCTION
116      00C9 45D8      117      0000H 42# 125 140      1      INCLUDE LEAF ACTIVAGE FROM DASH
117      00CA 45D9      118      0040H 50 58#      1      WITH LOAN FEEL
118      00CB 45DA      119      0097H 32# 77 79 115      1      SAVE SWAKE BEDW SMICH
119      00CC 45DB      120      0095H 30# 74 76 121      1      VMD ONLY TO LOAN SIGNIF
120      00CD 45DC      121      00A1H 34# 80 82 129      1      ADD IN SAVING FIGHT FUNCTION
121      00CE 45DD      122      0093H 23# 112 126      1      INCLUDE LEAF ACTIVAGE FROM DASH
122      00CF 45DE      123      0007H 43# 114 135      1      WITH LOAN FEEL
123      00D0 45DF      124      0090H 20 21 22 23 24 30 31 32 72      1      SAVE SWAKE BEDW SMICH
124      00D1 45E0      125      00A0H 33 34 35 37 73      1      VMD ONLY TO LOAN SIGNIF
125      00D2 45E1      126      0092H 22# 107      1      ADD IN SAVING FIGHT FUNCTION
126      00D3 45E2      127      00D0H 45 54 148      1      INCLUDE LEAF ACTIVAGE FROM DASH
127      00D4 45E3      128      00A0H 33# 86 88 136      1      WITH LOAN FEEL
128      00D5 45E4      129      0096H 31# 83 85 139      1      SAVE SWAKE BEDW SMICH
129      00D6 45E5      130      00A2H 35# 89 91 144      1      VMD ONLY TO LOAN SIGNIF
130      00D7 45E6      131      0094H 24# 133 141      1      ADD IN SAVING FIGHT FUNCTION
131      00D8 45E7      132      00A3H 37# 97      1      INCLUDE LEAF ACTIVAGE FROM DASH
132      00D9 45E8      133      0020H 41# 42 43 62 69 70 104 105 106      1      WITH LOAN FEEL
133      00DA 45E9      134      00B4H 75 78 81 84 87 90 96      1      SAVE SWAKE BEDW SMICH
134      00DB 45EA      135      00BFH 69 96 104#      1      VMD ONLY TO LOAN SIGNIF
135      00DC 45EB      136      008CH 53 59      1      ADD IN SAVING FIGHT FUNCTION
136      00DD 45EC      137      008AH 58      1      INCLUDE LEAF ACTIVAGE FROM DASH
137      00DE 45ED      138      0089H 60      1      WITH LOAN FEEL
138      00DF 45EE      139      008CH 65      1      SAVE SWAKE BEDW SMICH
139      00E0 45EF      140      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
140      00E1 45F0      141      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
141      00E2 45F1      142      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
142      00E3 45F2      143      0054H 55 69#      1      WITH LOAN FEEL
143      00E4 45F3      144      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
144      00E5 45F4      145      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
145      00E6 45F5      146      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
146      00E7 45F6      147      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
147      00E8 45F7      148      0054H 55 69#      1      WITH LOAN FEEL
148      00E9 45F8      149      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
149      00EA 45F9      150      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
150      00EB 45FA      151      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
151      00EC 45FB      152      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
152      00ED 45FC      153      0054H 55 69#      1      WITH LOAN FEEL
153      00EE 45FD      154      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
154      00EF 45FE      155      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
155      00F0 45FF      156      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
156      00F1 4600      157      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
157      00F2 4601      158      0054H 55 69#      1      WITH LOAN FEEL
158      00F3 4602      159      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
159      00F4 4603      160      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
160      00F5 4604      161      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
161      00F6 4605      162      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
162      00F7 4606      163      0054H 55 69#      1      WITH LOAN FEEL
163      00F8 4607      164      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
164      00F9 4608      165      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
165      00FA 4609      166      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
166      00FB 460A      167      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
167      00FC 460B      168      0054H 55 69#      1      WITH LOAN FEEL
168      00FD 460C      169      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
169      00FE 460D      170      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
170      00FF 460E      171      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
171      0100 460F      172      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
172      0101 4610      173      0054H 55 69#      1      WITH LOAN FEEL
173      0102 4611      174      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
174      0103 4612      175      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
175      0104 4613      176      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
176      0105 4614      177      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
177      0106 4615      178      0054H 55 69#      1      WITH LOAN FEEL
178      0107 4616      179      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
179      0108 4617      180      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
180      0109 4618      181      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
181      010A 4619      182      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
182      010B 461A      183      0054H 55 69#      1      WITH LOAN FEEL
183      010C 461B      184      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
184      010D 461C      185      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
185      010E 461D      186      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
186      010F 461E      187      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
187      0110 461F      188      0054H 55 69#      1      WITH LOAN FEEL
188      0111 4620      189      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
189      0112 4621      190      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
190      0113 4622      191      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
191      0114 4623      192      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
192      0115 4624      193      0054H 55 69#      1      WITH LOAN FEEL
193      0116 4625      194      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
194      0117 4626      195      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
195      0118 4627      196      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
196      0119 4628      197      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
197      011A 4629      198      0054H 55 69#      1      WITH LOAN FEEL
198      011B 462A      199      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
199      011C 462B      200      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
200      011D 462C      201      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
201      011E 462D      202      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
202      011F 462E      203      0054H 55 69#      1      WITH LOAN FEEL
203      0120 462F      204      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
204      0121 4630      205      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
205      0122 4631      206      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
206      0123 4632      207      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
207      0124 4633      208      0054H 55 69#      1      WITH LOAN FEEL
208      0125 4634      209      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
209      0126 4635      210      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
210      0127 4636      211      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
211      0128 4637      212      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
212      0129 4638      213      0054H 55 69#      1      WITH LOAN FEEL
213      012A 4639      214      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
214      012B 463A      215      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
215      012C 463B      216      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
216      012D 463C      217      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
217      012E 463D      218      0054H 55 69#      1      WITH LOAN FEEL
218      012F 463E      219      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
219      0130 463F      220      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
220      0131 4640      221      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
221      0132 4641      222      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
222      0133 4642      223      0054H 55 69#      1      WITH LOAN FEEL
223      0134 4643      224      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
224      0135 4644      225      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
225      0136 4645      226      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
226      0137 4646      227      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
227      0138 4647      228      0054H 55 69#      1      WITH LOAN FEEL
228      0139 4648      229      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
229      013A 4649      230      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
230      013B 464A      231      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
231      013C 464B      232      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
232      013D 464C      233      0054H 55 69#      1      WITH LOAN FEEL
233      013E 464D      234      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
234      013F 464E      235      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
235      0140 464F      236      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
236      0141 4650      237      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
237      0142 4651      238      0054H 55 69#      1      WITH LOAN FEEL
238      0143 4652      239      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
239      0144 4653      240      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
240      0145 4654      241      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
241      0146 4655      242      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
242      0147 4656      243      0054H 55 69#      1      WITH LOAN FEEL
243      0148 4657      244      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
244      0149 4658      245      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
245      014A 4659      246      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
246      014B 465A      247      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
247      014C 465B      248      0054H 55 69#      1      WITH LOAN FEEL
248      014D 465C      249      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
249      014E 465D      250      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
250      014F 465E      251      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
251      0150 465F      252      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
252      0151 4660      253      0054H 55 69#      1      WITH LOAN FEEL
253      0152 4661      254      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
254      0153 4662      255      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
255      0154 4663      256      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
256      0155 4664      257      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
257      0156 4665      258      0054H 55 69#      1      WITH LOAN FEEL
258      0157 4666      259      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
259      0158 4667      260      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
260      0159 4668      261      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
261      015A 4669      262      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
262      015B 466A      263      0054H 55 69#      1      WITH LOAN FEEL
263      015C 466B      264      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
264      015D 466C      265      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
265      015E 466D      266      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
266      015F 466E      267      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
267      0160 466F      268      0054H 55 69#      1      WITH LOAN FEEL
268      0161 4670      269      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
269      0162 4671      270      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
270      0163 4672      271      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
271      0164 4673      272      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
272      0165 4674      273      0054H 55 69#      1      WITH LOAN FEEL
273      0166 4675      274      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
274      0167 4676      275      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
275      0168 4677      276      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
276      0169 4678      277      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
277      016A 4679      278      0054H 55 69#      1      WITH LOAN FEEL
278      016B 467A      279      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
279      016C 467B      280      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
280      016D 467C      281      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
281      016E 467D      282      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
282      016F 467E      283      0054H 55 69#      1      WITH LOAN FEEL
283      0170 467F      284      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
284      0171 4680      285      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
285      0172 4681      286      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
286      0173 4682      287      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
287      0174 4683      288      0054H 55 69#      1      WITH LOAN FEEL
288      0175 4684      289      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
289      0176 4685      290      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
290      0177 4686      291      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
291      0178 4687      292      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
292      0179 4688      293      0054H 55 69#      1      WITH LOAN FEEL
293      017A 4689      294      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
294      017B 468A      295      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
295      017C 468B      296      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
296      017D 468C      297      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
297      017E 468D      298      0054H 55 69#      1      WITH LOAN FEEL
298      017F 468E      299      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
299      0180 468F      300      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
300      0181 4690      301      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
301      0182 4691      302      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
302      0183 4692      303      0054H 55 69#      1      WITH LOAN FEEL
303      0184 4693      304      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
304      0185 4694      305      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
305      0186 4695      306      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
306      0187 4696      307      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
307      0188 4697      308      0054H 55 69#      1      WITH LOAN FEEL
308      0189 4698      309      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
309      018A 4699      310      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
310      018B 469A      311      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
311      018C 469B      312      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
312      018D 469C      313      0054H 55 69#      1      WITH LOAN FEEL
313      018E 469D      314      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
314      018F 469E      315      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
315      0190 469F      316      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
316      0191 46A0      317      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
317      0192 46A1      318      0054H 55 69#      1      WITH LOAN FEEL
318      0193 46A2      319      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
319      0194 46A3      320      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
320      0195 46A4      321      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
321      0196 46A5      322      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
322      0197 46A6      323      0054H 55 69#      1      WITH LOAN FEEL
323      0198 46A7      324      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
324      0199 46A8      325      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
325      019A 46A9      326      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
326      019B 46AA      327      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
327      019C 46AB      328      0054H 55 69#      1      WITH LOAN FEEL
328      019D 46AC      329      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
329      019E 46AD      330      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
330      019F 46AE      331      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
331      01A0 46AF      332      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
332      01A1 46B0      333      0054H 55 69#      1      WITH LOAN FEEL
333      01A2 46B1      334      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
334      01A3 46B2      335      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
335      01A4 46B3      336      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
336      01A5 46B4      337      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
337      01A6 46B5      338      0054H 55 69#      1      WITH LOAN FEEL
338      01A7 46B6      339      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
339      01A8 46B7      340      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
340      01A9 46B8      341      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
341      01AA 46B9      342      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
342      01AB 46BA      343      0054H 55 69#      1      WITH LOAN FEEL
343      01AC 46BB      344      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
344      01AD 46BC      345      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
345      01AE 46BD      346      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
346      01AF 46BE      347      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
347      01B0 46BF      348      0054H 55 69#      1      WITH LOAN FEEL
348      01B1 46C0      349      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
349      01B2 46C1      350      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
350      01B3 46C2      351      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
351      01B4 46C3      352      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
352      01B5 46C4      353      0054H 55 69#      1      WITH LOAN FEEL
353      01B6 46C5      354      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
354      01B7 46C6      355      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
355      01B8 46C7      356      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
356      01B9 46C8      357      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
357      01BA 46C9      358      0054H 55 69#      1      WITH LOAN FEEL
358      01BB 46CA      359      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
359      01BC 46CB      360      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
360      01BD 46CC      361      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
361      01BE 46CD      362      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
362      01BF 46CE      363      0054H 55 69#      1      WITH LOAN FEEL
363      01C0 46CF      364      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
364      01C1 46D0      365      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
365      01C2 46D1      366      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
366      01C3 46D2      367      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
367      01C4 46D3      368      0054H 55 69#      1      WITH LOAN FEEL
368      01C5 46D4      369      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
369      01C6 46D5      370      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
370      01C7 46D6      371      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
371      01C8 46D7      372      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
372      01C9 46D8      373      0054H 55 69#      1      WITH LOAN FEEL
373      01CA 46D9      374      0054H 55 69#      1      SAVE SWAKE BEDW SMICH
374      01CB 46DA      375      0054H 55 69#      1      VMD ONLY TO LOAN SIGNIF
375      01CC 46DB      376      0054H 55 69#      1      ADD IN SAVING FIGHT FUNCTION
376      01CD 46DC      377      0054H 55 69#      1      INCLUDE LEAF ACTIVAGE FROM DASH
377      01CE 46DD      378      0054H 55 69#      
```

3

MCS-48 Family



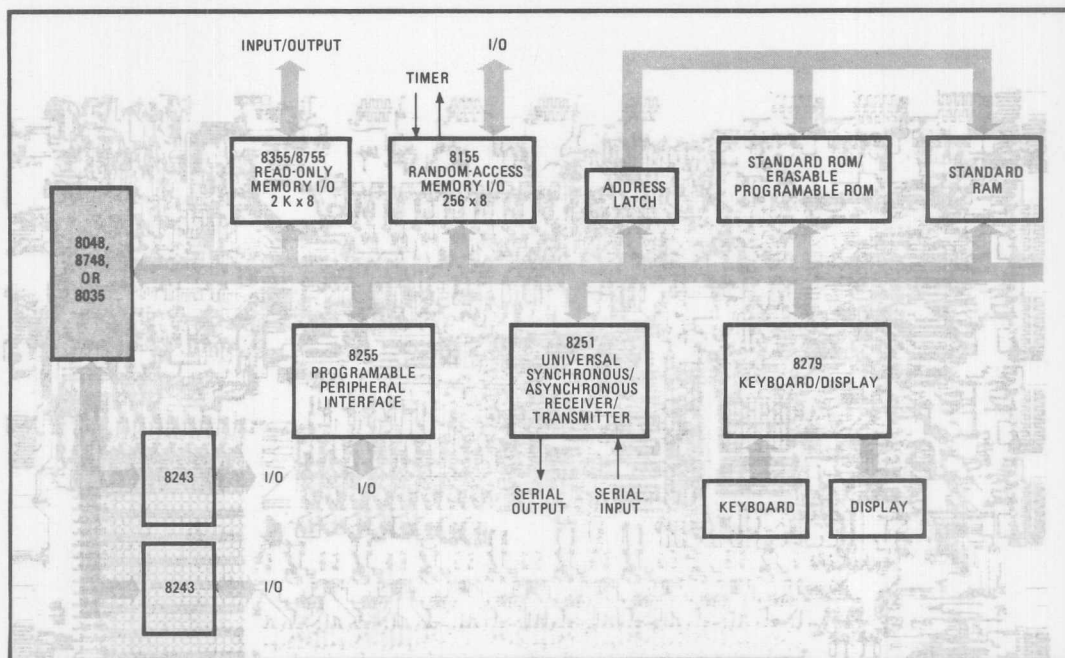
© Intel 1976

Single-chip 8-bit microcomputer fills gap between calculator types and powerful multichip processors

Capabilities range from stand-alone computing to high-power data processing; ultraviolet light erases programable ROM of one version

by Henry Blume, David Budde, Bill Morgan, Howard Raphael, Phil Salsbury, David Stamm,

Intel Corp., Santa Clara, Calif.



1. Expandable. Although well able to run a stand-alone controller by itself, the new processor can also work with other family members for larger control systems or with 8080 peripherals to handle complex data processing. This configuration typifies the MCS-48 capability.

□ Putting an 8-bit microcomputer onto a single chip is achievement enough, but realizing performance nearly equal to multiple-chip devices gives a bonus of added flexibility for the new family. The two devices that are the heart of the family are really high-performance, single-chip microcomputers that fill the gap between 4-bit calculator chips and the 8-bit multichip microprocessors. They can be used for the lowest levels of control, or, by being expanded with other ROM/RAM members of their family or with standard 8080 peripheral memory chips, they can be used in a wide range of high-powered data-processing systems.

The two versions of the microcomputer, the 8748 and the 8048, are like 4-bit calculator devices in that they each contain all the elements needed for stand-alone computing—central processing unit, program read-only memory, data random-access memory, input/output interface, plus clocks and timers. Yet they contain these elements in 8-bit configurations that vastly exceed the power of the calculator types and approach 8080 power.

Two ROM versions

The MCS-48 family is the first to offer a microprocessor with an erasable programmable ROM, which will prove handy for low-volume applications and those in which periodic update of the program memory is required. The family also has a CPU-only chip, the 8035, which can be used with external memories.

The 8748 has a 2708-type, 8,192-bit EPROM with a program that can be changed by clearing with ultraviolet light and reprogramming electrically in the usual way. The

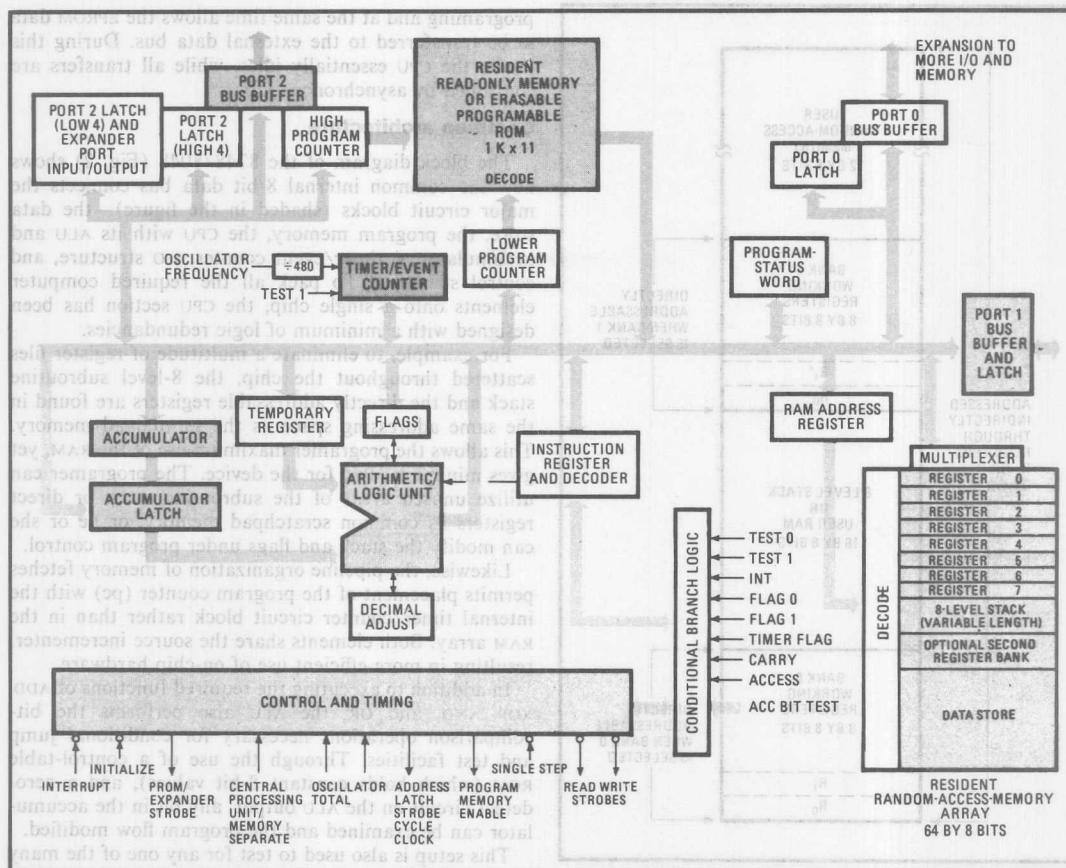
8048 has an 8-k mask-programable ROM. Together they give the user new flexibility: he can develop the program and build the prototypes with the reprogrammable chip and switch to mask ROMs for volume production.

The off-the-shelf 8748 also is perfect for quick-turnaround users who require small volumes only, since it can be programmed to meet any system specification in any quantity—in contrast to some single-chip controllers requiring mask programming at the factory, which is often available only in large quantities. Equally important, the 8748 can be used in control systems requiring periodic updating in the field, such as point-of-sale price-and-inventory controls. New program data can be fed into the system without a new ROM.

The free-standing operations of the 8048 and 8748 are made possible by the 1,024-by-8-bit ROM or EPROM for program memory, a 64-by-9-bit RAM for scratchpad functions, an 8-bit CPU consisting of an arithmetic/logic unit and accumulator for all the binary and decimal arithmetic functions, and an input/output facility that includes three 8-bit I/O ports plus three test/interrupt ports directly controlled by program instructions.

Memory and input/output of the processors can be expanded to handle large control applications (Fig. 1). There's an inexpensive expander chip, 8243, which allows the processor chips to handle an additional 16 I/O lines. Also included in the family are combination memory and I/O expanders, such as a 2,048-by-8-bit ROM with 16 I/O lines (8355), a 2-k-by-8-bit EPROM with 16 I/O lines (8755), and a 256-by-8-bit RAM with 22 I/O lines (8155).

The MCS-48 components also work directly with all



2. Stacked. The 8748 or 8048 processor chip supplies all the functions needed for a stand-alone microcomputer. It has a CPU complete with arithmetic/logic unit and accumulator, a 256-bit RAM, an 8,192-bit program ROM, a timer/event counter, and plenty of I/O capability.

the 8080 family of standard memory and peripheral parts, soon to number about 30 large-scale-integrated circuits. They include timers, programmable I/O controllers, universal synchronous/asynchronous receiver/transmitters, decoders, and keyboard/display controllers.

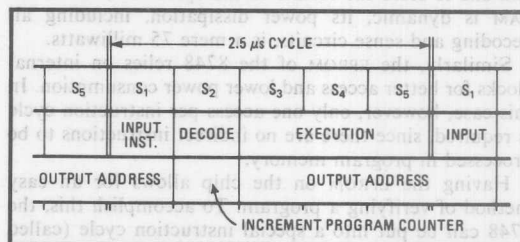
One-chip advantages

The integration of all the basic blocks of a microcomputer system into one circuit brings about some architectural advantages. When the device is used as a stand-alone controller, it need interface only with its I/O peripherals. This means that the execution speed of the processing is limited only by the speed of the chip, because there is no slowdown from transferring data between memory and CPU, as in multiple-chip designs.

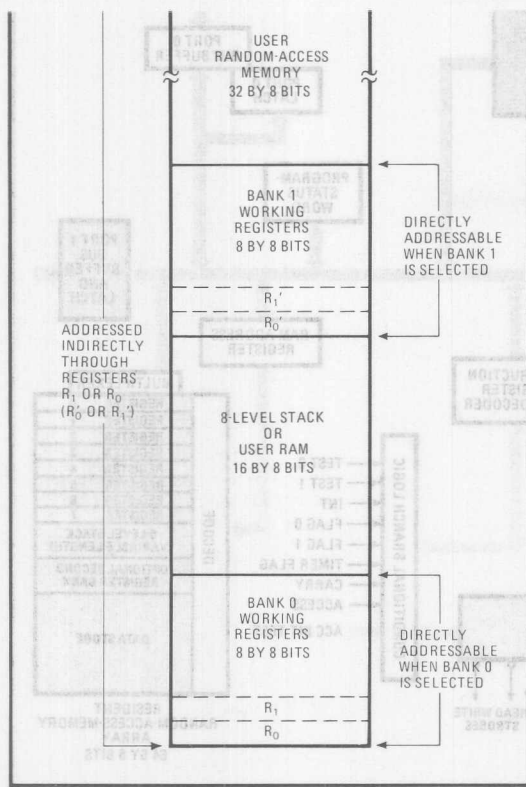
Moreover, technological upgrades can give enhanced performance without waiting for similar upgrades of external components, as is usually the case with multi-chip families. More immediately, the inclusion of data and program memories, which otherwise would have to be added separately to the system, simplifies the user's interface problems.

Having an active data store on the chip—the quasi-static 64-by-8-bit RAM—also simplifies system implementation, since all scratchpad operations simply became part of the CPU function. There is no need for refresh circuits operate the RAM; yet the device is dynamic in the sense that internal clocks are used for very fast, low-power access to the array.

The major objective was access to a RAM within a



3. Simple. Operating the 8748/8048 is extremely straightforward, with each 2.5-μs cycle consisting of five states. Instruction inputs are made in state 1, decoding and program incrementing in state 2. Program executions begin in state 3 and run through 4 and 5.



4. Powerful. The on-chip RAM, part of which is reserved for one or two banks of 8-bit working registers, also accommodates the stack of subroutine addresses, which can be eight levels deep. Each stack location can handle the program counter and status data.

fraction of an instruction cycle, so that those indirect internal instructions that require multiple addresses could still be executed in one instruction cycle. (Indirect RAM instructions require three separate accesses: one to fetch the address of the memory location to be operated on, one to fetch the contents of the addressed location, and one to store the results of the operation.) Since the RAM is dynamic, its power dissipation, including all decoding and sense circuits, is a mere 75 milliwatts.

Similarly, the EPROM of the 8748 relies on internal clocks for better access and lower power consumption. In this case, however, only one access per instruction cycle is required, since there are no indirect instructions to be processed in program memory.

Having the EPROM on the chip allows for an easy method of verifying a program. To accomplish this, the 8748 can be put into a special instruction cycle (called the third-state mode) for programing and verification of the EPROM. The CPU executes a special double-cycle instruction that allows the address and data information to be transferred to their respective registers during

controlled by asynchronous inputs.

Common architecture

The block diagram of the 8748/8048 (Fig. 2) shows how the common internal 8-bit data bus connects the major circuit blocks (shaded in the figure)—the data store, the program memory, the CPU with its ALU and accumulator, a timer/event counter, I/O structure, and control structure. To pack all the required computer elements onto a single chip, the CPU section has been designed with a minimum of logic redundancies.

For example, to eliminate a multitude of register files scattered throughout the chip, the 8-level subroutine stack and the directly addressable registers are found in the same addressing space as the scratchpad memory. This allows the programmer maximum use of the RAM, yet gives minimum logic for the device. The programmer can utilize unused areas of the subroutine stack or direct registers as common scratchpad memory, or he or she can modify the stack and flags under program control.

Likewise, the pipeline organization of memory fetches permits placement of the program counter (pc) with the internal timer/counter circuit block rather than in the RAM array. Both elements share the source incrementer, resulting in more efficient use of on-chip hardware.

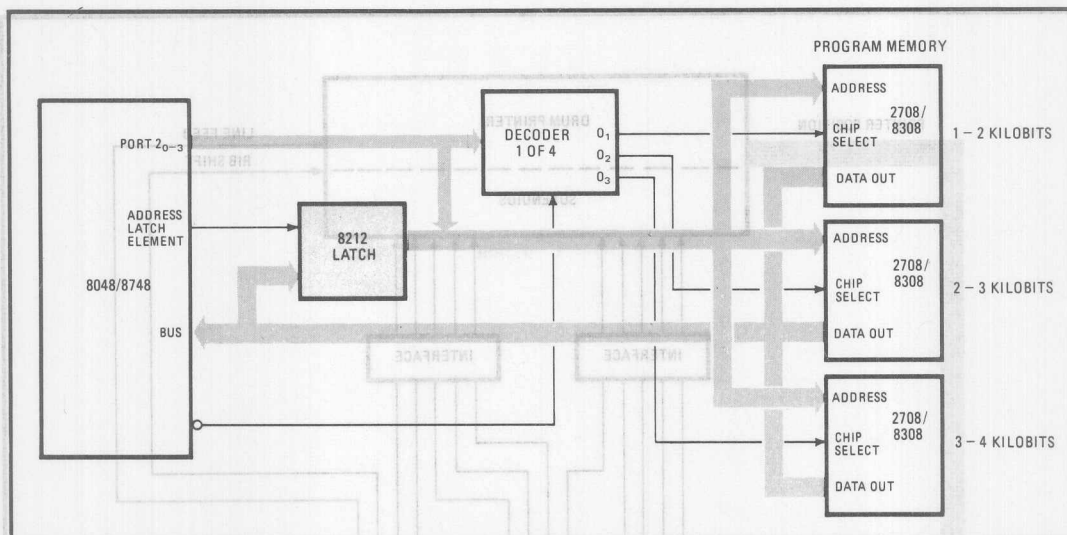
In addition to executing the required functions of ADD, XOR, AND, and OR, the ALU also performs the bit-comparison operations necessary for conditional jump and test facilities. Through the use of a control-table ROM (which holds constant 8-bit values), and a zero-detect circuit on the ALU output, any bit in the accumulator can be examined and the program flow modified.

This setup is also used to test for any one of the many conditional jumps. Each of the conditional-jump flags and inputs is sent to the ALU as an 8-bit conditional word and tested with the same circuitry used to examine individual accumulator bits.

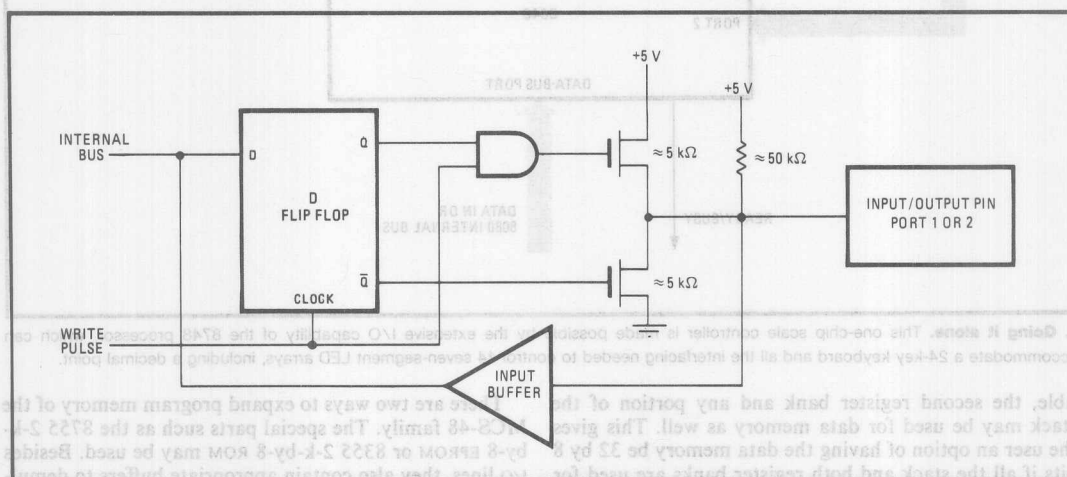
An internal oscillator also gives many system and device savings, such as the elimination of external components (except for a crystal or an RC network for setting the system's operating frequency). It also gives the chip designer maximum freedom in the structure of the internal clocking scheme, because there is no need for high-level, accurate clock inputs.

Through efficient use of internal bus transfers, most instructions can be executed in a single-cycle length. The exceptions are those instructions which require a second memory fetch or an external I/O transfer. In these cases, only a second cycle is required. Moreover, limiting instructions to two lengths reduces the complexity of the internal state generator. Since 70% of all instructions are executed in a single cycle, program-execution times and program-storage size are still minimized.

The multiplexed bus for address and data during external memory references maximizes the number of I/O pins available on a cost-effective 40-pin dual in-line package. For external program-memory references, bits of an additional I/O port are used for address lines, with the input/output data being restored after the memory



5. Latching on. Adding standard memories to the system is quickly done with external latch 8212, which allows standard memory parts to be hooked directly onto the 8748/8048 bus. Operation of the latch is under the control of signals from the processor.



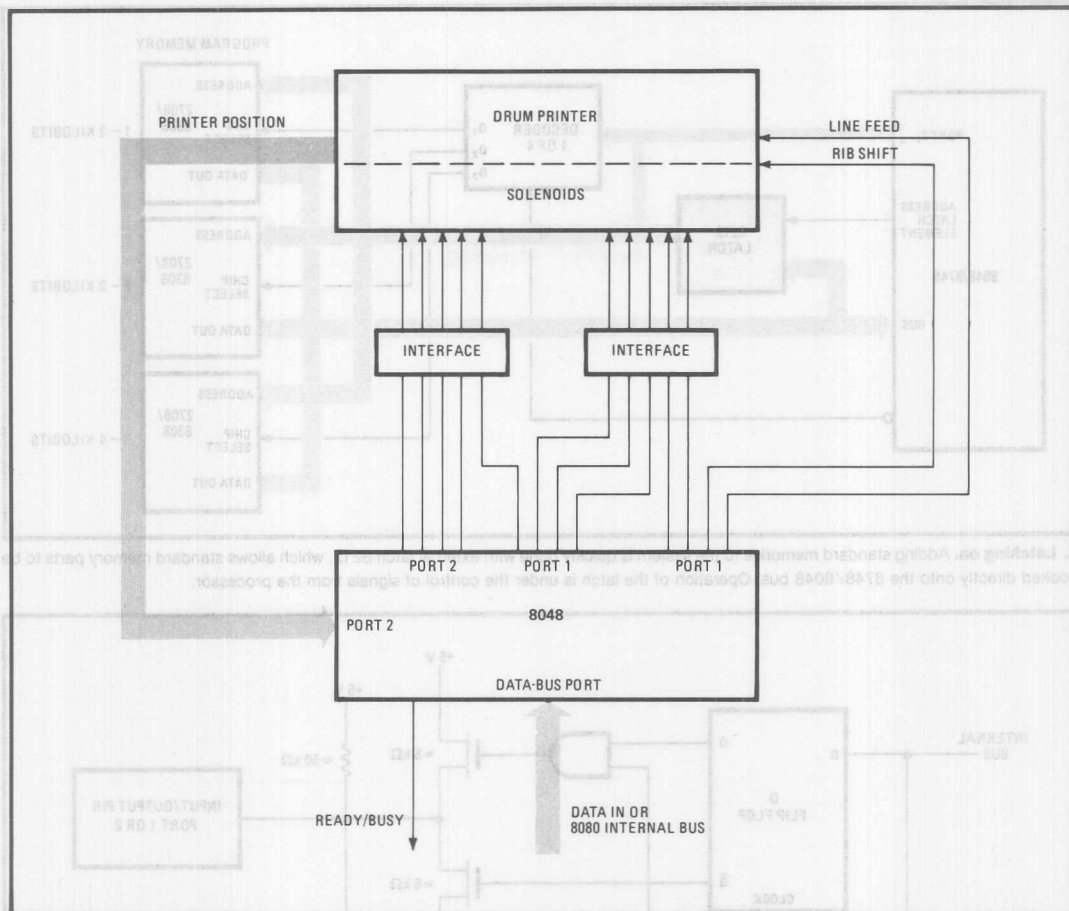
6. Mixing it up. Besides the main system port, 0, the processor chip has two others, 1 and 2, which allow inputs and outputs to be mixed on the same port. Here, writing a 0 causes the pull-down devices to sink the TTL load; writing a 1 calls on the 50-kilohm pull-up resistor.

One key to the simple operation of the 8748/8048 chip is the straightforward program sequences and timing needed for executing an instruction cycle (Fig. 3). Each cycle consists of five states. Instruction input is made in state 1, and decoding and pc incrementing is made in state 2. State 3 starts the beginning of the program execution, which can run through states 4 and 5. Simultaneously, the next cycle's program address is made in state 3, a pipelining (paralleling) of operations that increases device throughput significantly.

Because the chip is built with depletion-load silicon-gate n-channel technology, it operates off a single 5-volt supply with inputs and outputs that are compatible with

both transistor-transistor-logic and complementary metal-oxide-semiconductor devices. Instruction cycle time is a modest 2.5 microseconds and power consumption is a low 400 mW. Depletion-load techniques also pay off in practical chip sizes for volume production; the 8048 also is slightly over 200 mils on a side, while the 8748, with its big 8-k EPROM, is 221 by 261 mils. Storing data in the scratchpad is simple, because part of the RAM can be reserved for one or two banks of 8-bit working registers—eight registers per bank (Fig. 4). The scratchpad also contains the subroutine address stack, which can be eight levels deep. Each location can accommodate the 12-bit pc and 4-bit status data.

Since all locations in the stack are indirectly address-



7. Going it alone. This one-chip scale controller is made possible by the extensive I/O capability of the 8748 processor, which can accommodate a 24-key keyboard and all the interfacing needed to control 14 seven-segment LED arrays, including a decimal point.

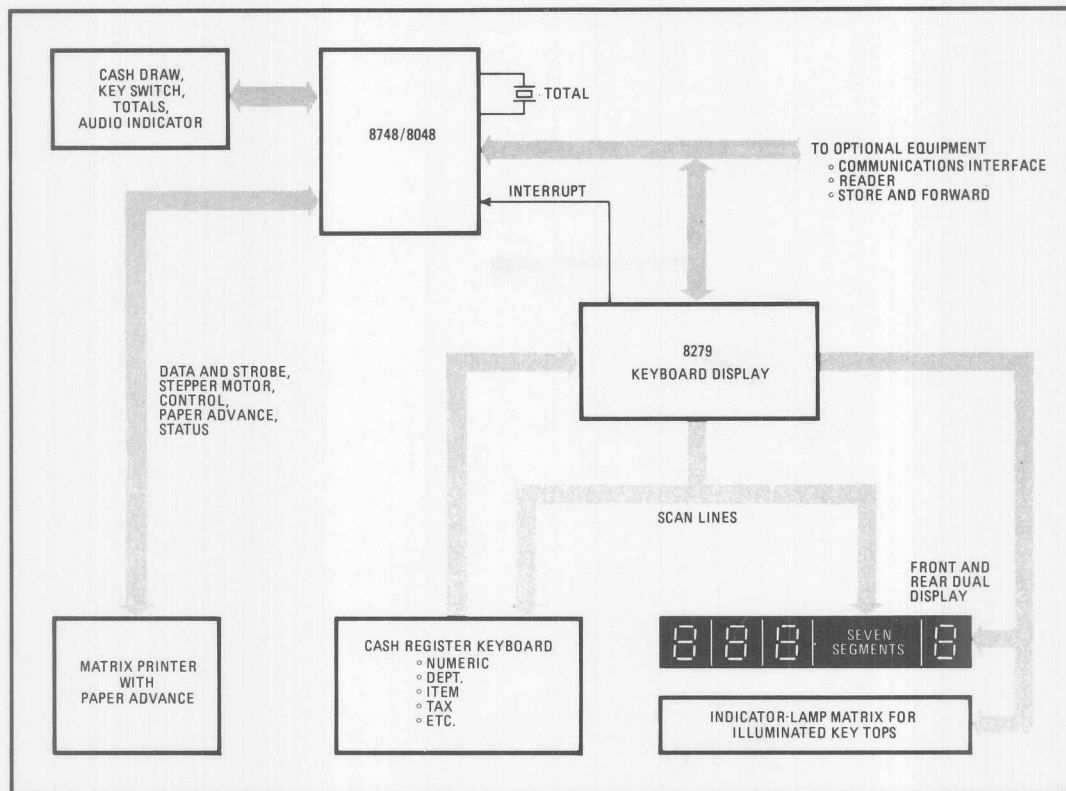
able, the second register bank and any portion of the stack may be used for data memory as well. This gives the user an option of having the data memory be 32 by 8 bits if all the stack and both register banks are used for program counting and status data, or 56 by 8 bits if only one register bank is used.

Program memory

The resident program memories on the 8048/8748 chips are handled so that they can be operated alone for programs of 1,024 bytes or less or combined with external ROM for expanded systems requiring larger programs. The program counter that feeds the memory is split into two parts. The low-order 8 bits can either address the resident 1-k ROM or be routed externally when addressing beyond 1,024 bits. (Since the 8035 contains no internal ROM, all address fetches are external.) The upper 4 bits of the program counter, located near port 2 (see Fig. 2), are gated out on that port for external reference. Two of these most significant 4 bits are then used for internal addressing requirements.

There are two ways to expand program memory of the MCS-48 family. The special parts such as the 8755 2-k-by-8 EPROM or 8355 2-k-by-8 ROM may be used. Besides I/O lines, they also contain appropriate buffers to demultiplex the 8-bit bus from the microcomputer chips to receive address and send back program-memory instructions. Alternately, standard memory parts, such as a 2708 EPROM or 8308 ROM may be used (Fig. 5). An external latch, such as the 8212, would latch up the address from the bus (via a signal from the 8048 or 8748) so that data could be returned on the bus. The high-order 4 bits of the address do not have to be latched, since they are not on the multiplexed bus.

The ALU, in conjunction with the accumulator, provides a full array of binary-and-decimal arithmetic, logic, shift, and increment/decrement functions. For example, the accumulator may be exchanged between registers, data memory, and program memory. Both the timer/counter and the program-status word are also accessible to the accumulator, through a latch that facilitates the accumulator source/destination instruc-



8. Working together. Proof of the MCS-48's ability to handle large systems is this gas-pump controller. The 8243 I/O expander chips allow the processor to interface with 47 lines and a USART communicating with a central control unit inside the service station.

tions. Here, the ALU generates a carry output fully accessible to the programmer under program control.

The timer/event counter is an 8-bit register that can operate in one of two modes, selectable under software control. As a timer, the device measures elapsed time. It is fed by the crystal frequency, divided by 280. At maximum frequency, the result is about 80 μ s per increment, or about 20 milliseconds over the counter range. As an event counter, a test line is designated to count 0 to 1 transitions of external events. As many as 256 transitions may be accommodated.

Both the timer and the counter indicate overflow by a maskable internal interrupt or by a testable flag bit. The internal interrupt may also be used to provide the system with a second external interrupt.

The input/output facilities of the 8048/8748 have been designed for maximum flexibility and expansion and are fully TTL-compatible. The basic facilities consist of three 8-bit I/O ports plus three test/interrupt inputs.

Port 0, called the bus, provides for system expansion. In essence, the port makes the bus completely compatible with an 8080 bus, so that all 8080 peripherals can be used with the MCS-48 family. In conjunction with four control and strobe lines, the port may be used for bidirectional interfacing to memories and I/O elements. For free-standing operations, it may be statically latched

or used as a general input port.

The remaining two I/O ports, 1 and 2, are termed "quasi-bidirectional" (Fig. 6). They allow inputs and outputs to be mixed on the same port. When writing a 0 (low value) to these ports, the pull-down device sinks the TTL load. When writing a 1, a large current is supplied through both pull-up devices to allow a fast transition. After a short time, they shut off and the pull-up of the 50-kilohm resistor sustains the 1 level.

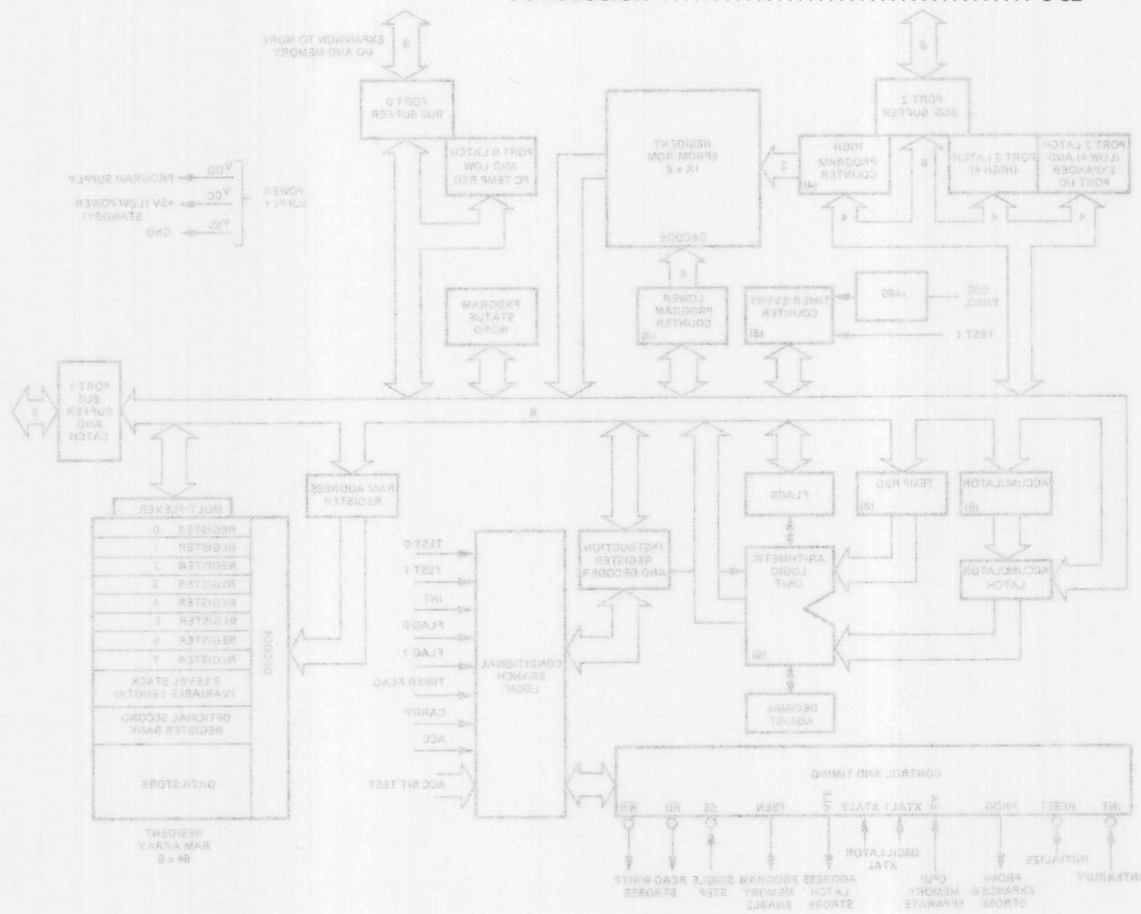
Applying the 8748/8048

Two applications show the range of complexity that can be accommodated with this family. Figure 7 shows a typical minimum-chip MCS-48 system, in this case, a drum printer controller. The three output ports allow the one-chip 8048 to control the printer position, ribbon shift, and line feed. Two interface drivers operate the solenoids.

Figure 8 shows a far more complex system, in which the MCS-48 implements a low-cost point-of-sale terminal. The I/O capability of the 8748/8048 chip can be expanded to control and monitor many cash-register operations. These might include cash in the drawer, key switch, totals, audio indicator, as well as matrix printer, cash-register keyboard, seven-segment display, and a variety of optional equipment. □

ADVANTAGE OF THE LATEST ADVANCES IN SILICON TECHNOLOGY	3-10
THE MCS-48 FAMILY	3-12
ANALOG I/O	3-16
TABLE LOOKUP TECHNIQUES	3-17
RECEIVING SERIAL CODES — BASIC APPROACHES	3-21
RECEIVING SERIAL CODE — A MORE SOPHISTICATED ALGORITHM	3-21
TRANSMITTING SERIAL CODE	3-31
GENERATING PARITY	3-32

CONCLUSION 3-32



INTRODUCTION

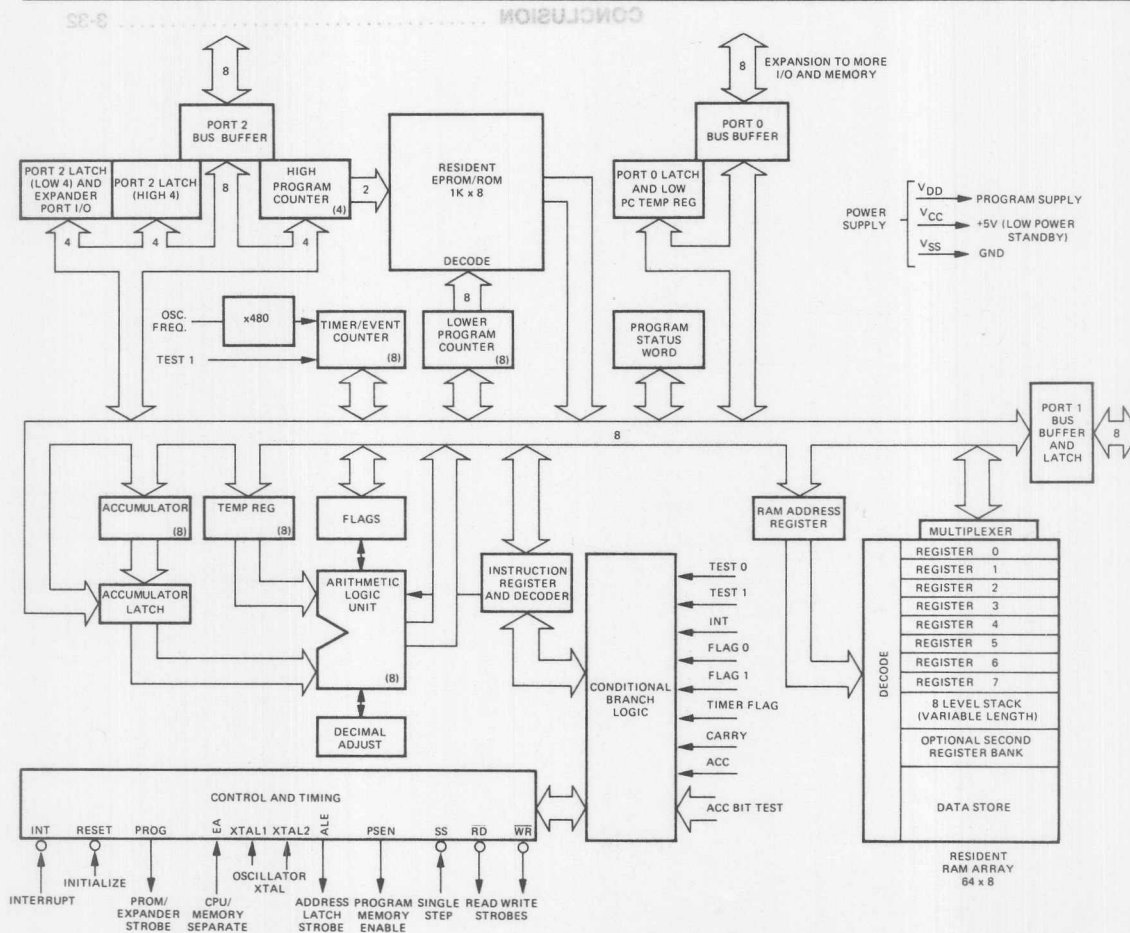
The INTEL® MCS-48™ family consists of a series of seven parts, including three processors, which take advantage of the latest advances in silicon technology to provide the system designer with an effective solution to a wide variety of design problems. The significant contribution of the MCS-48 family is that instead of consisting of integrated microcomputer components it consists of integrated microcomputer systems. A single integrated circuit contains the processor, RAM, ROM (or PROM), a timer, and I/O.

This application note suggests a variety of application techniques which are useful with the MCS-48. Rather than presenting the design of a complete system it describes the implementation of "subsystems" which are common to many micropro-

cessor based systems. The subsystems described are analog input and output, the use of tables for function evaluation, receiving serial code, transmitting serial code, and parity generation. After an overview of the MCS-48 family these areas are discussed in a more or less independent manner.

THE MCS-48™ FAMILY

The processors in the MCS-48 family all share an identical architecture. The only significant difference is the type of on board program storage which is provided. The 8748 (see Figure 1) includes 1024 bytes of erasable, programmable, ROM (EPROM), the 8048 replaces the EPROM with an equivalent amount of mask programmed ROM, and the 8035 provides the CPU function with no on board program storage. All three of these processors



MCS-48™ Internal Structure

INSTRUCTION SET

Mnemonic	Description	Bytes	Cycle	Mnemonic	Description	Bytes	Cycles
Accumulator	ADD A,R	1	1	Subroutine	CALL	2	2
	ADD A,@R	1	1		RET	1	2
	ADD A,#data	2	2		RETR	1	2
	ADDC A,R	1	1				
	ADDC A,@R	1	1	Flags	CLR C	1	1
	ADDC A,#data	2	2		CPL C	1	1
	ANL A,R	1	1		CLR F0	1	1
	ANL A,@R	1	1		CPL F0	1	1
	ANL A,#data	2	2		CLR F1	1	1
	ORL A,R	1	1		CPL F1	1	1
	ORL A,@R	1	1	Data Movers	MOV A,R	1	1
	ORL A,#data	2	2		MOV A,@R	1	1
	XRL A,R	1	1		MOV A,#data	2	2
	XRL A,@R	1	1		MOV R,A	1	1
	XRL A,#data	2	2		MOV @R,A	1	1
	INC A	1	1		MOV R,#data	2	2
	DEC A	1	1		MOV @R,#data	2	2
	CLR A	1	1		MOV A,PSW	1	1
	CPL A	1	1		MOV PSW,A	1	1
	DA A	1	1		XCH A,R	1	1
	SWAP A	1	1		XCH A,@R	1	1
	RL A	1	1		XCHD A,@R	1	1
	RLC A	1	1		MOVX A,@R	1	2
	RR A	1	1		MOVX @R,A	1	2
	RRC A	1	1		MOVP A,@A	1	2
					MOVP3 A,@A	1	2
Input/Output	IN A,P	1	2	Timer/Counter	MOV A,T	1	1
	OUTL P,A	1	2		MOV T,A	1	1
	ANL P,#data	2	2		STRT T	1	1
	ORL P,#data	2	2		STRT CNT	1	1
	INS A,BUS	1	2		STOP TCNT	1	1
	OUTL BUS,A	1	2		EN TCNTI	1	1
	ANL BUS,#data	2	2		DIS TCNTI	1	1
	ORL BUS,#data	2	2	Control	EN I	1	1
	MOVD A,P	1	2		DIS I	1	1
	MOVD P,A	1	2		SEL RB0	1	1
	ANLD P,A	1	2		SEL RB1	1	1
	ORLD P,A	1	2		SEL MB0	1	1
					SEL MB1	1	1
Registers	INC R	1	1		ENTO CLK	1	1
	INC @R	1	1	NOP			
	DEC R	1	1				
	JMP addr	2	2				
	JMPP @A	1	2				
	DJNZ R,addr	2	2				
	JC addr	2	2				
	JNC addr	2	2				
	JZ addr	2	2				
	JNZ addr	2	2				
Branch	JTO addr	2	2				
	JNT0 addr	2	2				
	JT1 addr	2	2				
	JNT1 addr	2	2				
	JF0 addr	2	2				
	JF1 addr	2	2				
	JTF addr	2	2				
	JNI addr	2	2				
	JBB addr	2	2				

Figure 2. 8048/8748/8035 Instruction Set

When installed in a system only the 5-volt supply is needed. Aside from program storage, these chips include 64 bytes of data storage (RAM), an eight bit timer which can also be used to count external events, 27 programmable I/O pins and the processor itself. The processor offers a wide range of instruction capability including many designed for bit, nibble, and byte manipulation. The instruction set is summarized in Figure 2.

Aside from the processors, the MCS-48 family includes 4 devices: one pure I/O device and 3 combination memory and I/O devices. The pure I/O device is the 8243, a device which is connected to a special 4 bit bus provided by the MCS-48 processors and which provides 16 I/O pins which can be programmatically controlled.

The combination memory and I/O devices consist of the 8355, the 8755, and the 8155. The 8355 and the 8755 both provide 2,048 bytes of program storage and two eight bit data ports. The only difference between these devices is that the 8355 contains masked program ROM and the 8755 contains EPROM. The 8155 combines 256 bytes of data storage (RAM), two eight bit data ports, a six bit control port, and a 14 bit programmable timer.

Figure 3 shows the various system configurations which can be achieved using the MCS-48 family of parts. It should also be noted that eight of the processors' I/O lines have been configured as a bidirectional bus which can be used to interface to standard Intel peripheral parts such as the 8251 USART (for serial I/O), the 8255A PPI (provides 24 I/O lines) and the complete range of memory components.

More detailed information concerning the MCS-48 family can be obtained from the "MCS-48 Microcomputer User's Manual" which provides a complete description of the MCS-48 family and its members. A general familiarity with this document will make the application techniques which follow easier to understand.

ANALOG I/O

If analog I/O is required for a MCS-48™ system there are many alternatives available from the makers of analog I/O modules. By searching through their catalogs it is possible to find almost any combination of features which is technically feasible. Perhaps the best example of such modules are the MP-10 and MP-20 hybrid modules recently introduced by Burr-Brown Research Corporation. The MP-10 provides two analog outputs and the MP-20 provides 16 analog inputs. Both of these units were

DATA MEMORY (RAM)	1K	8048 4-8155 [5] (101)	8035 8355 4-8155 [5] (116)	8048 8355 4-8155 [5] (116)	8035 2-8355 4-8155 [5] (131)
	832				
	768	8048 3-8155 [4] (80)	8035 8355 3-8155 [4] (95)	8048 8355 3-8155 [4] (95)	8035 2-8355 3-8155 [4] (110)
	578				
	512	8048 2-8155 [3] (59)	8035 8355 2-8155 [3] (74)	8048 8355 2-8155 [3] (74)	8035 2-8355 2-8155 [3] (89)
	320				
256		8048 8155 [2] (38)	8035 8355 8155 [2] (53)	8048 8355 8155 [2] (53)	8035 2-8355 8155 [2] (68)
	64	8048 [1] (24)	8035 8355 [1] (28)	8048 8355 [1] (28)	8035 2-8355 [1] (43)
		1K	2K	3K	4K
		PROGRAM MEMORY (ROM)			

Figure 3. The Expanded MCS-48™ System

specifically designed to interface with microprocessors.

A block diagram of the MP-10 is shown in Figure 4. It consists of two eight bit digital to analog converters, two eight bit latches which are loaded from the data bus, and address decoding logic to determine when the latches should be loaded. The D/A converters each generate an analog output in the range of 10 volts with an output impedance of 1Ω. Accuracy is ±0.4% of full scale and the output is stable 25μsec after the eight bit binary data is loaded into the appropriate latch. The latches are loaded by the write pulse (WR) whenever the proper address is presented to the MP-10. The lower two addresses (A₀ and A₁) are used internally by the device. Addresses A₂ & A₃ are compared with the address determination inputs B₂ and B₃. If their signals are found to be equal, and if addresses A₄-A₁₃ are all high, then the device is selected and one of the latches will be loaded. Address bit A₁ selects between output 1 and output 2. If address bit A₀ is set then the initialization channel of the DIA is selected. In order to prepare for operation a data pattern of 80H must

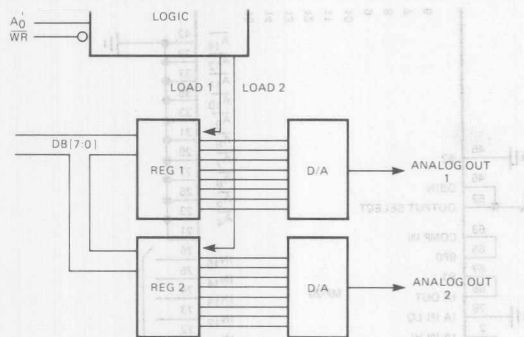


Figure 4. MP-10 Block Diagram

be output to this channel following the reset of the device.

A block diagram of the MP-20 analog to digital converter is shown in figure 5. This unit consists of a 16 input analog multiplexer, an instrumentation amplifier, an eight bit successive approximation analog to digital converter, and control logic. The 16 input multiplexer can be used to input either 16 single ended or 8 differential inputs. The output from the multiplexer is fed into the instrumentation amplifier which is configured so that it can easily be strapped for single ended 0-5 volt inputs, single ended ± 5 volt inputs, or differential 0-5 volt signals. Provisions are made for an external gain control resistor on the amplifier. The gain control equation is:

$$G = 2 + \frac{50k\Omega}{R_{ext}}$$

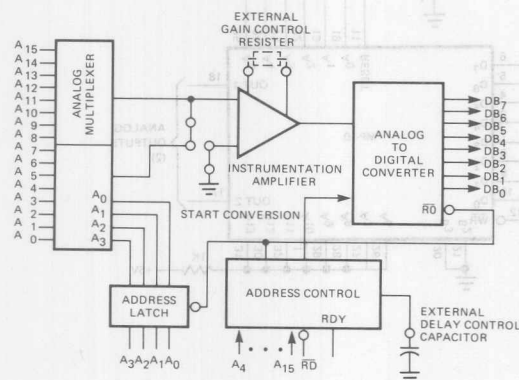


Figure 5. MP-20 Analog Subsystem

gauges can be accommodated. The output from the amplifier is applied to the actual A/D converter which provides an eight bit output with guaranteed monotonicity and an accuracy of $\pm 0.4\%$ of full scale. Note that this accuracy is specified for the entire module, not just for the converter itself. The control logic monitors address lines A15 through A4 to determine when the address of the unit has been selected. An address that the unit will respond to is determined by 11 address control pins, labeled $\overline{A4}$ through $\overline{A14}$. If one of these pins is tied to a logic 0 then the corresponding address pin must be high in order for the unit to be selected. If the pin is tied to a logic 1 then the corresponding address pin must be low. If the address of the module is selected when \overline{MEMR} pulse occurs, the lower four addresses (A3-A0) are stored in a latch which addresses the multiplexer. The coincidence of the proper address and \overline{MEMR} also initiates a conversion and gates the output of the converter on to the eight bit data bus.

The control logic of the MP-20 was designed to operate directly with an MCS-80TM system. When a \overline{MEMR} occurs and a conversion is initiated the MP-20 generates a READY signal which is used to extend the cycle of the 8080A for the duration of the conversion. READY is brought high after the conversion is complete which allows the 8080A to initiate a conversion and read the resulting data in a single, albeit long, memory or I/O cycle. The conversion time of the MP-20 depends on the gain selected for the amplifier. With no external resistor ($R = \infty$) the gain is two and the conversion time is 35 μsec . For $R = 510\Omega$ the gain is:

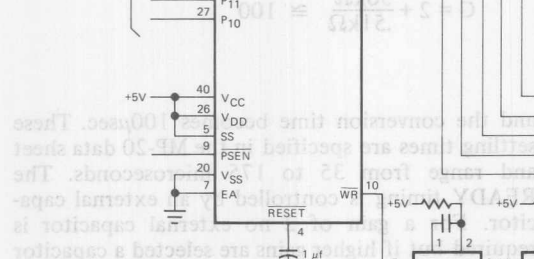
$$G = 2 + \frac{50k\Omega}{.51k\Omega} \cong 100$$

and the conversion time becomes 100 μsec . These settling times are specified in the MP-20 data sheet and range from 35 to 175 microseconds. The READY timing is controlled by an external capacitor. For a gain of 2 no external capacitor is required but if higher gains are selected a capacitor is needed to extend the timing.

A schematic showing both the MP-10 D/A and the MP-20 A/D connected to the 8748 is shown in Figure 6. This configuration, which consists of only four major components, gives an excellent example of what modern technology can do for

The diagram illustrates the timing of the 8080A microprocessor's UNCOMMITTED I/O PINS relative to the INT and MCS-48™ signals. The signals are shown as a sequence of pulses, with the UNCOMMITTED I/O PINS signal being a series of pulses that occur during the MCS-48™ signal's high period. The signals are labeled as follows:

- UNCOMMITTED I/O PINS
- INT
- MCS-48™
- P20
- P21
- P22
- P23
- P24
- P25
- P26
- P27



- An eight bit microprocessor
- 64 bytes of RAM
- 1024 bytes of UV erasable PROM
- A timer/event counter
- 16 digital I/O pins
- 2 testable input pins
- An interrupt capability
- 16 eight bit analog inputs
- 2 eight bit analog outputs

In order to provide reset capability for the analog devices without dedicating an I/O pin from the MCS-48, special addresses are used as reset channels. Executing any MOVX with an address of 0XXXXXXX will reset the A/D module; a similar operation with an address of X1XXXXXX will reset the D/A; a MOVX with an address of 01XXXXXX will reset both devices. All data transfers are accomplished with the upper two bits of the address field equal to 10. A summary of the addressing of the analog devices is shown in Table 1. Notice that except for an initialization channel for the D/A (which must

INPUT OR OUTPUT	
0 X X X X X X X X	Reset A/D
X 1 X X X X X X	Reset D/A
INPUT	
0 0 1 1 n n n n	Read A/D Channel n n n n
OUTPUT	
1 0 1 1 0 0 0 1	Initialize D/A
1 0 1 1 0 0 0 0	Write Channel 1
1 0 1 1 0 0 1 0	Write Channel 2

As was mentioned previously, the MP-20 was designed to use the READY line of the 8080A. Obviously this presents a problem since the MCS-48 does not support a READY line (with its attendant requirement of entering WAIT state). The necessity of a READY input can be overcome by performing a read operation to set the channel address, waiting the required delay (35 μ sec for a gain of two) and then performing a second read to actually obtain the data. The second read will read in the data from the channel selected by the first read irrespective of the channel selected for the second read. Thus it is possible to use the second read to set up the channel for the third read. Each read can read in the current channel and select the next channel for conversion.

LOC	OBJ	SEQ	SOURCE STATEMENT
		0	
		1	
		2	-----
		3	TEST PROGRAM FOR ANALOG OUTPUT
		4	THIS PROGRAM OUTPUTS A SAW-
		5	TOOTH WAVEFORM BY OUTPUTTING
		6	AN INCREMENTING PATTERN.
		7	-----
		8	
		9	
		10	EQUATES
		11	-----
		12	
0003		13	INITCH EQU 0B3H ; D/A INITIALIZATION CHANNEL
0008		14	INITDD EQU 0B4H ; D/A INITIALIZATION DATA
000B		15	DATCH EQU 0BBH ; D/A DATA CHANNEL
		16	
		17	-----
		18	START OF TEST
		19	-----
0100		20	
		21	ORG 100H ; INITIALIZE D/A
0100 2300		22	START: MOV A, #INITD
0102 0B03		23	MOV R0, #INITCH
0104 90		24	MOVX @R0, A
		25	; TEST LOOP-OUTPUT SAWTOOTH
0105 0B00		26	LOOP: MOV R0, #DATCH
0107 17		27	INC A
0109 00 90		28	MOVX @R0, A
0109 2405		29	LOOP
		30	; END OF PROGRAM
		31	END

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Figure 7b. A/D Exercise Program

TABLE LOOKUP TECHNIQUES

In the previous section the interface between analog I/O devices and the MCS-48™ was discussed. In many applications involving analog I/O one quickly finds that nature is inherently nonlinear, and the mathematics involved in 'linearizing it' can tax the computational power of the microprocessor, particularly if it has other tasks to perform. Problems of this nature are good candidates for the use of tables.

As an example of how tables can be used as part of an analog output scheme, consider a system which requires an MCS-48 to output a variable frequency sinusoidal waveform. One method of performing this function would be to use the timer to generate an interrupt at a fixed rate of 256 times the desired output frequency. At each interrupt the appropriate value of the sine function could be calculated from the MacLaurin series:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \dots \frac{(-1)^k x^{2k+1}}{(2k+1)!}$$

Where K is chosen to be large enough to provide the required accuracy.

All mnemonics copyrighted © Intel Corporation 1976.

severely limit the possible output frequencies which could be obtained. As an alternative to calculating these values in real time, the values could be precalculated off line and stored in a table. Upon each interrupt the MCS-48 would merely have to retrieve the appropriate value from the table and output it to the D/A converter. the MCS-48 provides a special instruction which can be used to access data in a table. If the table is stored in the last 256 bytes of the first kilobyte of MCS-48 memory then the table lookup can be performed by loading the independent variable (time in this case) into the accumulator and executing the instruction.

MOV3 A, @A

This instruction uses the initial contents of the accumulator to index into page 3 of program storage. The location pointed to is read and the contents placed in the accumulator. If (as is often the case) a table of fewer than 256 entries is required, then the table can be located in any page of program memory and the instruction:

MOV3 A, @A

can be used to retrieve data from the table. This instruction operates in the same manner as does the previous instruction except that the current page of program storage is assumed to contain the table.

If it is possible to devote slightly more of the microprocessor's time to the table look up process, then a much smaller table can often be utilized by taking advantage of interpolation to determine values of the function between values which are actual entries in the table. As an example of this

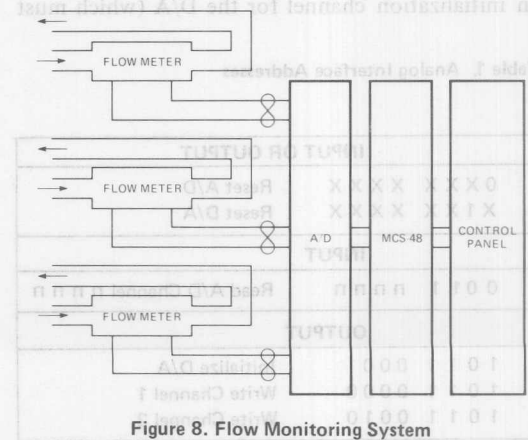


Figure 8. Flow Monitoring System

the flow through the three pipes, add them, and display the total flow on the control panel. The system consists of three flow meters which generate a differential voltage which is some function of flow, an A/D system with at least three differential inputs, an MCS-48, and a control panel. The schematic shown in Figure 6 could easily become part of this system, with the spare digital I/O of the MCS-48 used as an interface to the control panel. The simplicity of this system is clouded by the flow transducers, which are assumed to be not only nonlinear but also to require individual calibration (this is not an unreasonable assumption for a flow transducer). By using a table look up process and an 8748 the flow transducers can be calibrated and the results of the calibration tests stored directly in tables in the 8748. (The 8748 has a PROM in place of the ROM of the 8048 and thus makes such 'one off' programming practical.)

The results which might be obtained from calibrating one of the flow meters is shown in Figure 9. The results are plotted as gals/hour versus the measured voltage generated by the transducer. The voltage is shown in hexadecimal form so that it corresponds directly to the digital output of the analog to digital converter. The flow required to generate seventeen evenly spaced voltages (0H-100H in steps of 10H) has been measured and plotted. This information is shown in tabular form in Figure 10. It is necessary to generate a program which will convert any measured input from 00H to FFH into the flow in units which can be interpreted by a human operator. This can easily be done by simple interpolation.

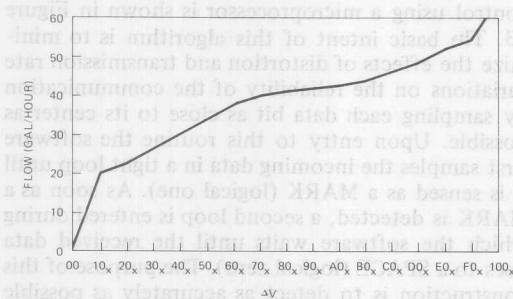


Figure 9. Flow Calibration Curve

TRANSDUCER VOLTAGE (HEX)	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0	100
MEASURED FLOW (GAL/HOUR)	0	10	22	26	30	34	38	40	41	42	43	45	48	49	53	56	63

Figure 10. Tabulated Flow Data

ificant four bits (/4) will be used to retrieve one of the table values. The lower four bits (3-0) will be used to interpolate between this value and the value retrieved from the next higher location in the table. If the upper four bits are given the symbol I and the lower four bits the symbol N, then the interpolation can be expressed as:

$$F(x) = F(I) + \frac{N}{16} [F(I+1) - F(I)]$$

Where x is the measured voltage and F(x) is the corresponding flow.

If, as an example, the transducer voltage was measured as 48H then the flow (ref. Figure 10) would be:

$$F = 30 + \frac{8}{16} (34 - 30) = 32$$

A subroutine which implements this calculation is shown in Figure 11. Before it is called the independent variable (V) is placed in the accumulator and register R1 is set to point at the first value in the table. Aside from simple additions and subtractions the only arithmetic required is to multiply two values and then divide them by 16. The multiplication is handled via a subroutine which is also shown in Figure 11. The division by 16 can be performed by a four place right shift followed by a rounding operation. The routine shown will handle a monotonic increasing function of a single independent variable. Fairly simple modifications are required for nonmonotonic functions. Functions of two variables can be handled by interpolating on a plane rather than along a straight line. Although this is more time consuming, requiring an interpolation for each of the independent variables and a third to interpolate the final answer, it still provides a simple means of quickly calculating the required function. The use of tables can offer a powerful technique for function evaluation to the designer.

RECEIVING SERIAL CODE—BASIC APPROACHES

Many microprocessor based systems require some form of serial communication. Serial communication is extensively used because it allows two or more pieces of equipment to exchange information with a minimal number of interconnecting wires. The minimization of interconnecting wires results in simpler, cheaper, interconnects because fewer (or smaller) cables and connectors are required. Since the required number of drivers and receivers required is reduced, it can become economically feasible to provide much higher noise immunity

```

LOC  OBJ  SEQ  SOURCE STATEMENT
0 ; *****
1 ;
2 ; APPROX
3 ; AT ENTRY R1 POINTSAT TABLE
4 ; A HAS INDEPENDENT VARIABLE
5 ;
6 ; *****
7 ;
8 ;
9 ; EQUATES
10 ;
11 ;
12 RX0 EQU R0 ; POINTER 0
13 RX1 EQU R1 ; POINTER 1
14 AEX EQU R2 ; EXTENSION OF A REGISTER
15 COUNT EQU R3 ; COUNTER
16 TEMP EQU R4 ; TEMP STORAGE
17 ;
18 ;
19 ; APPROXIMATION
20 ;
21 ;
22 DRG 100H ; POINT RX0 AT TEMP
23 ;
24 APPROX: MOV RX0, #TEMP ; TEMP=N AND 0FH
25 ; ; A=P AND 0FH
26 ;
27 MOV @RX0, #0
28 XCHD A, @RX0
29 SWAP A
30 ;
31 ADD A, RX1 ; RX1=BASE+A
32 MOV RX1, A
33 ;
34 ; RX1=TABLE(P)
35 ; A=TABLE(P+1)
36 MOVFP3 A, @A
37 XCH A, RX1
38 INC A
39 MOVFP3 A, @A
40 ; A=TABLE (P+1)-TABLE(P)
41 CPL A
42 ADD A, RX1
43 CPL A
44 ; A=N*A/16
45 CALL MULT
46 MOV RX0, #AEX
47 XCHD A, @RX0
48 SWAP A
49 XCH A, AEX
50 JB3 ADJUST
51 ADJUST: XCH A, AEX
52 INC A
53 ; A=A*TABLE(P)
54 ADD A, RX1
55 ; RETURN

```

Figure 11. Table Lookup With Interpolation

with more sophisticated (and expensive) line terminators. The final, and usually most persuasive, argument in favor of serial communication is that it may be the only method available to accomplish the job. The obvious example of this is telecommunications where it is necessary to encode parallel information into serial format in order to communicate via the telephone network. The intent of this section is to show how the facilities of the MCS-48™ can be brought to bear on the problem of serial communication.

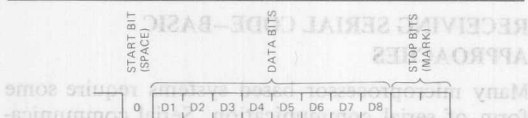


Figure 12. Serial ASCII Code

Probably the most common form of serial communication is that used by the ubiquitous Teletype—serial ASCII. This format, shown in Figure 12, consists of a START bit (0 or SPACE) followed by eight data bits which are in turn followed by two STOP bits (1 or MARK). In actual practice the

```

LOC  OBJ  SEQ  SOURCE STATEMENT
011C B3 56 RET
57
58
59 ;
60 ; MULTIPLY
61 ;
62 ; SET UP COUNT AND AEX
63 MULT: MOV COUNT, #8
64 MOV AEX, #0
65 ; CLEAR CARRY
66 LOOFA: CLR C
67 ; IF MULTIPLIER (0) < 1 THEN SHIFT PRODUCT
68 ;
69 ;
70 ;
71 ;
72 ;
73 ; LOOP UNTIL DONE
74 DJNZ COUNT, LOOFA
75 RET
76 ; ELSE ADD MULTIPLIER AND SHIFT PRODUCT
77 SSUM: XCH A, AEX
78 ADD A, @RX0
79 RRC A
80 XCH A, AEX
81 RRC A
82 ; LOOP UNTIL DONE
83 DJNZ COUNT, LOOFA
84 RET
85
86 ;
87 ;
88 ; TABLE TO TEST PROGRAM
89 ;
90
91 DRG 380H
92
93 ;
94 ; THIS TABLE IS FROM FIG 10
95 ;
96 ;
97 ;
98 ;
99 ;
100 ;
101 ;
102 ;
103 ;
104 ;
105 ;
106 ;
107 ;
108 ;
109 ;
110 ;
111 ;
112 ;
113 ;
114 ;
115 ;
116 ;
117 ;
118 ;
119 ;
120 ;
121 ;
122 ;
123 ;
124 ;
125 ;
126 ;
127 ;
128 ;
129 ;
130 ;
131 ;
132 ;
133 ;
134 ;
135 ;
136 ;
137 ;
138 ;
139 ;
140 ;
141 ;
142 ;
143 ;
144 ;
145 ;
146 ;
147 ;
148 ;
149 ;
150 ;
151 ;
152 ;
153 ;
154 ;
155 ;
156 ;
157 ;
158 ;
159 ;
160 ;
161 ;
162 ;
163 ;
164 ;
165 ;
166 ;
167 ;
168 ;
169 ;
170 ;
171 ;
172 ;
173 ;
174 ;
175 ;
176 ;
177 ;
178 ;
179 ;
180 ;
181 ;
182 ;
183 ;
184 ;
185 ;
186 ;
187 ;
188 ;
189 ;
190 ;
191 ;
192 ;
193 ;
194 ;
195 ;
196 ;
197 ;
198 ;
199 ;
200 ;
201 ;
202 ;
203 ;
204 ;
205 ;
206 ;
207 ;
208 ;
209 ;
210 ;
211 ;
212 ;
213 ;
214 ;
215 ;
216 ;
217 ;
218 ;
219 ;
220 ;
221 ;
222 ;
223 ;
224 ;
225 ;
226 ;
227 ;
228 ;
229 ;
230 ;
231 ;
232 ;
233 ;
234 ;
235 ;
236 ;
237 ;
238 ;
239 ;
240 ;
241 ;
242 ;
243 ;
244 ;
245 ;
246 ;
247 ;
248 ;
249 ;
250 ;
251 ;
252 ;
253 ;
254 ;
255 ;
256 ;
257 ;
258 ;
259 ;
260 ;
261 ;
262 ;
263 ;
264 ;
265 ;
266 ;
267 ;
268 ;
269 ;
270 ;
271 ;
272 ;
273 ;
274 ;
275 ;
276 ;
277 ;
278 ;
279 ;
280 ;
281 ;
282 ;
283 ;
284 ;
285 ;
286 ;
287 ;
288 ;
289 ;
290 ;
291 ;
292 ;
293 ;
294 ;
295 ;
296 ;
297 ;
298 ;
299 ;
300 ;
301 ;
302 ;
303 ;
304 ;
305 ;
306 ;
307 ;
308 ;
309 ;
310 ;
311 ;
312 ;
313 ;
314 ;
315 ;
316 ;
317 ;
318 ;
319 ;
320 ;
321 ;
322 ;
323 ;
324 ;
325 ;
326 ;
327 ;
328 ;
329 ;
330 ;
331 ;
332 ;
333 ;
334 ;
335 ;
336 ;
337 ;
338 ;
339 ;
340 ;
341 ;
342 ;
343 ;
344 ;
345 ;
346 ;
347 ;
348 ;
349 ;
350 ;
351 ;
352 ;
353 ;
354 ;
355 ;
356 ;
357 ;
358 ;
359 ;
360 ;
361 ;
362 ;
363 ;
364 ;
365 ;
366 ;
367 ;
368 ;
369 ;
370 ;
371 ;
372 ;
373 ;
374 ;
375 ;
376 ;
377 ;
378 ;
379 ;
380 ;
381 ;
382 ;
383 ;
384 ;
385 ;
386 ;
387 ;
388 ;
389 ;
390 ;
391 ;
392 ;
393 ;
394 ;
395 ;
396 ;
397 ;
398 ;
399 ;
400 ;
401 ;
402 ;
403 ;
404 ;
405 ;
406 ;
407 ;
408 ;
409 ;
410 ;
411 ;
412 ;
413 ;
414 ;
415 ;
416 ;
417 ;
418 ;
419 ;
420 ;
421 ;
422 ;
423 ;
424 ;
425 ;
426 ;
427 ;
428 ;
429 ;
430 ;
431 ;
432 ;
433 ;
434 ;
435 ;
436 ;
437 ;
438 ;
439 ;
440 ;
441 ;
442 ;
443 ;
444 ;
445 ;
446 ;
447 ;
448 ;
449 ;
450 ;
451 ;
452 ;
453 ;
454 ;
455 ;
456 ;
457 ;
458 ;
459 ;
460 ;
461 ;
462 ;
463 ;
464 ;
465 ;
466 ;
467 ;
468 ;
469 ;
470 ;
471 ;
472 ;
473 ;
474 ;
475 ;
476 ;
477 ;
478 ;
479 ;
480 ;
481 ;
482 ;
483 ;
484 ;
485 ;
486 ;
487 ;
488 ;
489 ;
490 ;
491 ;
492 ;
493 ;
494 ;
495 ;
496 ;
497 ;
498 ;
499 ;
500 ;
501 ;
502 ;
503 ;
504 ;
505 ;
506 ;
507 ;
508 ;
509 ;
510 ;
511 ;
512 ;
513 ;
514 ;
515 ;
516 ;
517 ;
518 ;
519 ;
520 ;
521 ;
522 ;
523 ;
524 ;
525 ;
526 ;
527 ;
528 ;
529 ;
530 ;
531 ;
532 ;
533 ;
534 ;
535 ;
536 ;
537 ;
538 ;
539 ;
540 ;
541 ;
542 ;
543 ;
544 ;
545 ;
546 ;
547 ;
548 ;
549 ;
550 ;
551 ;
552 ;
553 ;
554 ;
555 ;
556 ;
557 ;
558 ;
559 ;
560 ;
561 ;
562 ;
563 ;
564 ;
565 ;
566 ;
567 ;
568 ;
569 ;
570 ;
571 ;
572 ;
573 ;
574 ;
575 ;
576 ;
577 ;
578 ;
579 ;
580 ;
581 ;
582 ;
583 ;
584 ;
585 ;
586 ;
587 ;
588 ;
589 ;
590 ;
591 ;
592 ;
593 ;
594 ;
595 ;
596 ;
597 ;
598 ;
599 ;
600 ;
601 ;
602 ;
603 ;
604 ;
605 ;
606 ;
607 ;
608 ;
609 ;
610 ;
611 ;
612 ;
613 ;
614 ;
615 ;
616 ;
617 ;
618 ;
619 ;
620 ;
621 ;
622 ;
623 ;
624 ;
625 ;
626 ;
627 ;
628 ;
629 ;
630 ;
631 ;
632 ;
633 ;
634 ;
635 ;
636 ;
637 ;
638 ;
639 ;
640 ;
641 ;
642 ;
643 ;
644 ;
645 ;
646 ;
647 ;
648 ;
649 ;
650 ;
651 ;
652 ;
653 ;
654 ;
655 ;
656 ;
657 ;
658 ;
659 ;
660 ;
661 ;
662 ;
663 ;
664 ;
665 ;
666 ;
667 ;
668 ;
669 ;
670 ;
671 ;
672 ;
673 ;
674 ;
675 ;
676 ;
677 ;
678 ;
679 ;
680 ;
681 ;
682 ;
683 ;
684 ;
685 ;
686 ;
687 ;
688 ;
689 ;
690 ;
691 ;
692 ;
693 ;
694 ;
695 ;
696 ;
697 ;
698 ;
699 ;
700 ;
701 ;
702 ;
703 ;
704 ;
705 ;
706 ;
707 ;
708 ;
709 ;
710 ;
711 ;
712 ;
713 ;
714 ;
715 ;
716 ;
717 ;
718 ;
719 ;
720 ;
721 ;
722 ;
723 ;
724 ;
725 ;
726 ;
727 ;
728 ;
729 ;
730 ;
731 ;
732 ;
733 ;
734 ;
735 ;
736 ;
737 ;
738 ;
739 ;
740 ;
741 ;
742 ;
743 ;
744 ;
745 ;
746 ;
747 ;
748 ;
749 ;
750 ;
751 ;
752 ;
753 ;
754 ;
755 ;
756 ;
757 ;
758 ;
759 ;
760 ;
761 ;
762 ;
763 ;
764 ;
765 ;
766 ;
767 ;
768 ;
769 ;
770 ;
771 ;
772 ;
773 ;
774 ;
775 ;
776 ;
777 ;
778 ;
779 ;
780 ;
781 ;
782 ;
783 ;
784 ;
785 ;
786 ;
787 ;
788 ;
789 ;
790 ;
791 ;
792 ;
793 ;
794 ;
795 ;
796 ;
797 ;
798 ;
799 ;
800 ;
801 ;
802 ;
803 ;
804 ;
805 ;
806 ;
807 ;
808 ;
809 ;
810 ;
811 ;
812 ;
813 ;
814 ;
815 ;
816 ;
817 ;
818 ;
819 ;
820 ;
821 ;
822 ;
823 ;
824 ;
825 ;
826 ;
827 ;
828 ;
829 ;
830 ;
831 ;
832 ;
833 ;
834 ;
835 ;
836 ;
837 ;
838 ;
839 ;
840 ;
841 ;
842 ;
843 ;
844 ;
845 ;
846 ;
847 ;
848 ;
849 ;
850 ;
851 ;
852 ;
853 ;
854 ;
855 ;
856 ;
857 ;
858 ;
859 ;
860 ;
861 ;
862 ;
863 ;
864 ;
865 ;
866 ;
867 ;
868 ;
869 ;
870 ;
871 ;
872 ;
873 ;
874 ;
875 ;
876 ;
877 ;
878 ;
879 ;
880 ;
881 ;
882 ;
883 ;
884 ;
885 ;
886 ;
887 ;
888 ;
889 ;
890 ;
891 ;
892 ;
893 ;
894 ;
895 ;
896 ;
897 ;
898 ;
899 ;
900 ;
901 ;
902 ;
903 ;
904 ;
905 ;
906 ;
907 ;
908 ;
909 ;
910 ;
911 ;
912 ;
913 ;
914 ;
915 ;
916 ;
917 ;
918 ;
919 ;
920 ;
921 ;
922 ;
923 ;
924 ;
925 ;
926 ;
927 ;
928 ;
929 ;
930 ;
931 ;
932 ;
933 ;
934 ;
935 ;
936 ;
937 ;
938 ;
939 ;
940 ;
941 ;
942 ;
943 ;
944 ;
945 ;
946 ;
947 ;
948 ;
949 ;
950 ;
951 ;
952 ;
953 ;
954 ;
955 ;
956 ;
957 ;
958 ;
959 ;
960 ;
961 ;
962 ;
963 ;
964 ;
965 ;
966 ;
967 ;
968 ;
969 ;
970 ;
971 ;
972 ;
973 ;
974 ;
975 ;
976 ;
977 ;
978 ;
979 ;
980 ;
981 ;
982 ;
983 ;
984 ;
985 ;
986 ;
987 ;
988 ;
989 ;
990 ;
991 ;
992 ;
993 ;
994 ;
995 ;
996 ;
997 ;
998 ;
999 ;
1000 ;

```

eight data bit usually consists of even parity on the remaining seven data bits; for the purposes of this discussion the eighth bit will be considered only as data. A minor variation of this format deletes one of the STOP bits. An algorithm which might be used to sample serial data under software control using a microprocessor is shown in Figure 13. The basic intent of this algorithm is to minimize the effects of distortion and transmission rate variations on the reliability of the communication by sampling each data bit as close to its center as possible. Upon entry to this routine the software first samples the incoming data in a tight loop until it is sensed as a MARK (logical one). As soon as a MARK is detected, a second loop is entered during which the software waits until the received data goes to a SPACE (logical zero). The purpose of this construction is to detect as accurately as possible the leading edge of the START bit. This instant of time will be used as a reference point for sampling all of the following bits in the character. After sensing the leading edge of the START bit a wait of one half the expected bit time is implemented. The period of the incoming signal is called P for convenience. At the end of this wait the serial line is tested—if it is MARK then the START bit was

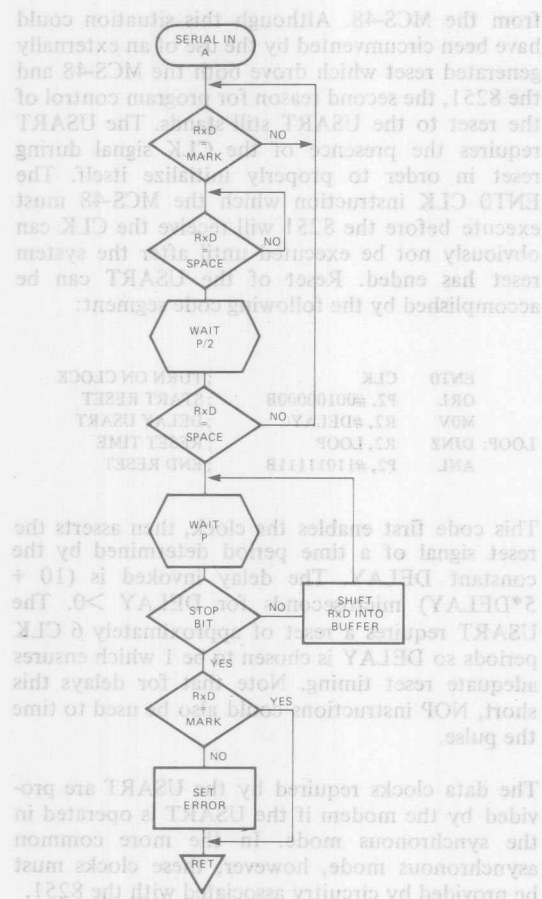


Figure 13. Sample Serial Input Routine

The data clocks required by the USART are provided by the microcontroller. The 8251 USART is an asynchronous mode, however, the clocks must be provided by circuitry associated with the 8251. The 2.9904 MHz crystal was chosen because the resulting frequency divided to provide internal and receive clocks to the USART. Assuming the USART is in invalid and the process is reinitialized. If the line is still a SPACE, then the START bit is assumed to be valid and a delay of one bit time is started. At the completion of the delay the first data bit is sampled and a new delay of one bit time is initiated. This process is repeated until all eight data bits have been sampled. The last bit sampled is checked to determine if it is a valid STOP bit (a MARK). If it is, the character is assumed to be valid; if it is not, the character has a framing error and is probably invalid. A listing of a program which implements the above procedure is shown in Figure 14.

A disadvantage of the approach outlined in Figure 13 is that while the processor is inputting data serially it must totally dedicate itself to this task. Accurate timing can only be maintained if the program remains in a tight wait loop without allowing itself to be diverted to other functions. During reception of a character from a Teletype

the processor will spend only a 100μsecs or so processing data and the rest of the 100 milliseconds waiting to do the processing at the right time. This lack of efficiency (approximately 0.1%) in the utilization of processing power is why devices such as the 8251 USART find broad application in micro-processor systems.

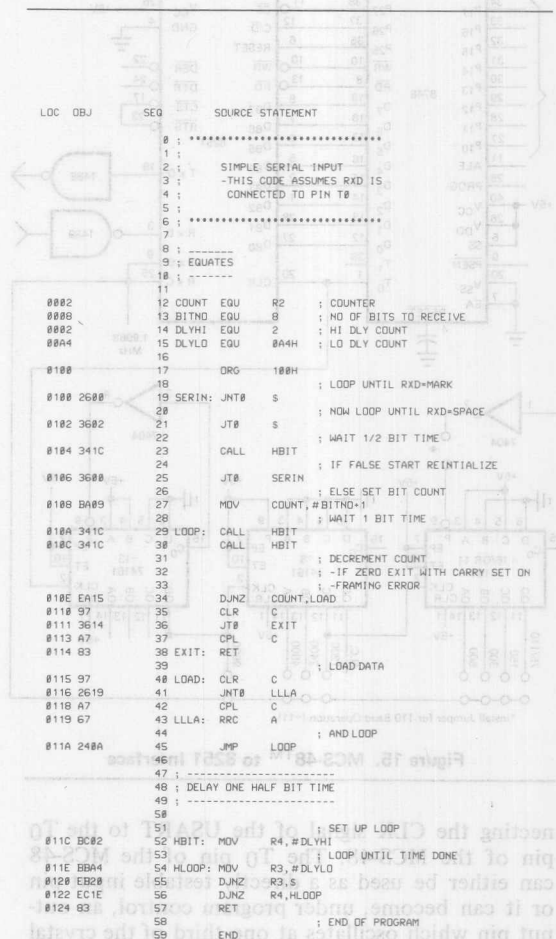


Figure 14. Simple Serial Input

The 8251 USART is simple to interface to the MSC-48. Figure 15 shows such an interface. The USART requires a high speed clock (CLK), an initialization signal (RESET), data clocks (TxC and RxC), and data in order to operate. A circuit showing the connection of an 8748 to an 8251 USART is shown in Figure 15. In the circuit shown the high speed clock (which is used for internal sequencing by the USART) is provided by con-

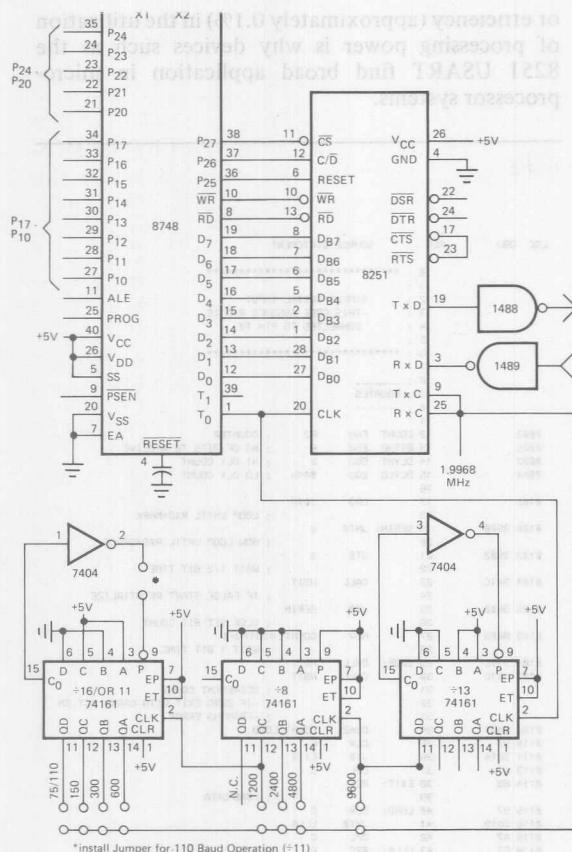


Figure 15. MCS-48™ to 8251 Interface

necting the CLK signal of the USART to the T0 pin of the MCS-48. The T0 pin of the MCS-48 can either be used as a directly testable input pin or it can become, under program control, an output pin which oscillates at one third of the crystal frequency. (Note that once this pin is designated by the software to be an output it will remain so until the system is reset.) In Figure 15 the crystal frequency is 5.9904 MHz so the clock provided to the 8251 is 1.9968 MHz, which conforms to its specifications.

The initialization signal to the USART (RESET) is provided programmatically by manipulation of bit 5 of port 2. It was necessary to place the reset of the 8251 under program control for two reasons. The first reason is that the MCS-48 does not supply a reset signal to other devices. The reason for this is that it was felt to be more useful to provide another pin of I/O function instead of a RESET OUT signal

the 8251, the second reason for program control of the reset to the USART still stands. The USART requires the presence of the CLK signal during reset in order to properly initialize itself. The ENT0 CLK instruction which the MCS-48 must execute before the 8251 will receive the CLK can obviously not be executed until after the system reset has ended. Reset of the USART can be accomplished by the following code segment:

```

        ENT0    CLK            ;TURN ON CLOCK
        ORL     P2, #00100000B ;START RESET
        MOV     R2, #DELAY     ;DELAY USART
LOOP:    DJNZ   R2, LOOP        ;RESET TIME
        ANL     P2, #11011111B ;END RESET

```

This code first enables the clock, then asserts the reset signal of a time period determined by the constant DELAY. The delay invoked is (10 + 5*DELAY) microseconds for DELAY > 0. The USART requires a reset of approximately 6 CLK periods so DELAY is chosen to be 1 which ensures adequate reset timing. Note that for delays this short, NOP instructions could also be used to time the pulse.

The data clocks required by the USART are provided by the modem if the USART is operated in the synchronous mode. In the more common asynchronous mode, however, these clocks must be provided by circuitry associated with the 8251.

The 5.9904 MHz crystal was chosen because the resulting 1.9968 MHz clock to the USART can be evenly divided to provide transmit and receive clocks to the USART. Assuming the USART is in the x16 mode (i.e. it requires data clocks 16 times the baud rate) the 1.9968 MHz signal can be divided by 13 to generate the proper clock rate for 9600 baud operation. This 9600 baud clock can be further divided to give 4800, 2400, 1200, 600, and 300 baud signals. The 1200 baud signal can be divided by 11 to give a 109.1 baud signal which is within 1% of the 110 baud required by Teletypes.

The MCS-48 communicates with the 8251 in a memory mapped mode (i.e. as if the 8251 were external RAM). The instructions available to do this are MOVX ∂R_j , A which stores the contents of the accumulator at the external RAM location addressed by R_j ($j=0$ or 1), and its complement, the MOVX A, @ R_j instruction which moves data from the external RAM into the accumulator. Since the MCS-48 multiplexes addresses and data on the same eight bit bus an external latch would be required in order to address the USART with

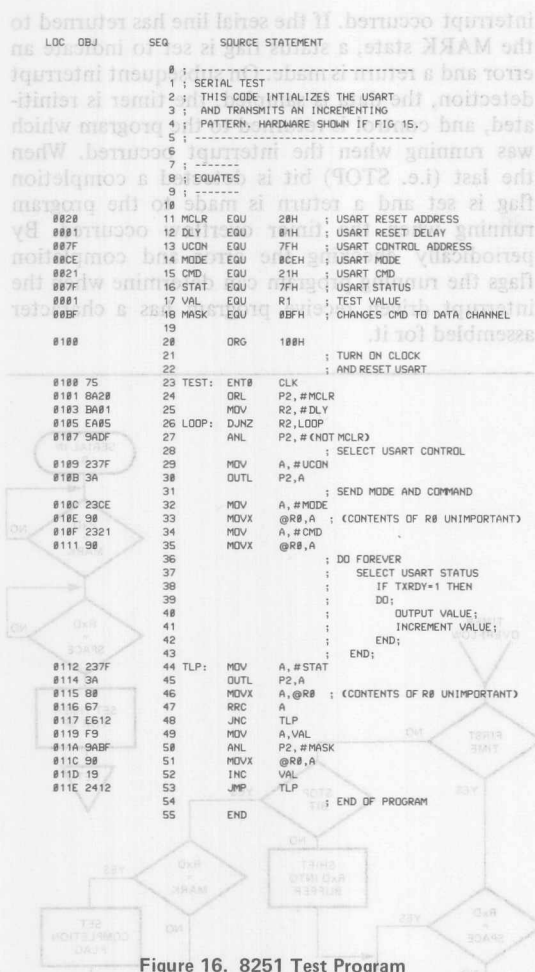


Figure 16. 8251 Test Program

R0 or R1. In order to minimize the circuitry in Figure 15 an approach utilizing some of the I/O pins of the MCS-48 to address the 8251 was chosen instead. By connecting the chip select (\overline{CS}) input of the 8251 to bit 7 of port 2 (P27) and similarly connecting the $\overline{C/D}$ address line of the 8251 to bit 6 of port 2 (P26) it is possible to address the 8251 without using R0 or R1. The instruction sequence to access the 8251 is to first reset P27 and set P26 to the appropriate state, use a MOVX instruction to perform the appropriate operation, and then finally set P27 to deselect the 8251. As a concrete example of this addressing, Figure 16 shows the code necessary to initialize the 8251 and output an incrementing test pattern on a status driven basis. If more than one 8251 were to be added to the MCS-48, or if other types of peripheral circuitry would be required (e.g. an 8253 timer to generate the data clocks) it would probably become desirable

to add the circuitry necessary to use R0 or R1 to address the peripheral devices. The circuitry which has to be added to Figure 15 in order to make use of R0 or R1 to address the USART is shown in Figure 17. Note that only the changes to Figure 15 are shown. The additional component required is the 8212 eight bit latch. This latch is loaded, whenever a valid address is on the bus by the Address Latch Enable (ALE) signal provided by the MCS-48. During an external read or write cycle this address is used to address the 8251 in a linear select mode. In the circuit shown, the 8251 will be selected by any address with bit 1 a logical zero (XXXXXX0X) and the selection of control or data transfer ($\overline{C/D}$) will be based on bit zero of the address obtained from R0 or R1. Figure 18 shows the program of Figure 16 modified to utilize the addressing inherent in the MOVX instructions.

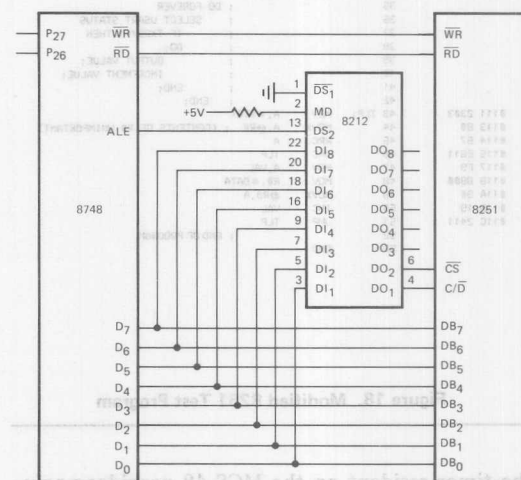


Figure 17. Modified MCS-48 to 8251 Interface

RECEIVING SERIAL CODE—A MORE SOPHISTICATED ALGORITHM

Although the USART does an admirable job of performing the serial I/O function for the MCS-48TM, there are some situations where it can not be used. These situations may be caused by economic factors, such as an extremely cost sensitive design, or because the code which must be utilized cannot be accommodated by the USART. An example of of such a code will be discussed later. Recall that the principal objection to the approach to serial input shown in Figure 13 was that it consumes much of the processor's power by merely spinning in loops in order to wait preset time delays.


```

LOC OBJ SEQ SOURCE STATEMENT
0 :-----
1 : SERIAL TEST
2 : THIS CODE INITIALIZES THE USART
3 : AND TRANSMITS AN INCREMENTING
4 : PATTERN. HARDWARE SHOWN IF FIG 17.
5 :-----
6 :
7 :
8 : EQUATES
9 :-----
10 :
11 MCLR EQU 20H : USART RESET ADDRESS
12 DLY EQU 81H : USART RESET DELAY
13 UCON EQU 83H : USART CONTROL ADDRESS
14 MODE EQU 8CEH : USART MODE
15 CMD EQU 21H : USART CMD
16 STAT EQU 83H : USART STATUS
17 VAL EQU R1 : TEST VALUE
18 DATA EQU 00 : USART DATA ADDRESS
19 :
20 ORG 100H
21 : TURN ON CLOCK
22 : AND RESET USART
23 TEST: ENT0 CLK
24 DRL P2, #MCLR
25 MOV R2, #DLY
26 DJNZ R2, LOOP
27 ANL P2, #NOT MCLR
28 : SELECT USART CONTROL
29 MOV A, #UCON
30 : SEND MODE AND COMMAND
31 MOV A, #MODE
32 MOVX @R0, A : (CONTENTS OF R0 UNIMPORTANT)
33 MOV A, #CMD
34 MOVX @R0, A
35 : DO FOREVER
36 : SELECT USART STATUS
37 IF TXRDY+1 THEN
38 DO
39 : OUTPUT VALUE;
40 INCREMENT VALUE;
41 END;
42 : END;
43 TLP: MOV A, #STAT
44 MOVX @R0, A : (CONTENTS OF R0 UNIMPORTANT)
45 RRC A
46 JNC TLP
47 MOV A, VAL
48 MOV R0, #DATA
49 MOVX @R0, A
50 INC VAL
51 JMP TLP
52 : END OF PROGRAM
53 END
1111 2303
1113 08
1114 67
1115 E611
1117 F9
1118 B000
111A 98
111B 19
111C 2411

```

Figure 18. Modified 8251 Test Program

The timer resident on the MCS-48 provides a solution to this problem. Instead of spinning in a loop the program can set the timer for a given interval, start it, and proceed to other tasks. When the timer overflows, an interrupt will be generated to notify the software that the present time period has elapsed. An extension of the algorithm of Figure 13 which uses the timer in this fashion is shown in Figure 19. This algorithm is identical to the preceding one up until the detection of the leading edge of the start bit. At this point the timer is set to one half of the bit time (P) and a return is made to the calling program which can start additional processing. At the completion of this time interval a timer overflow interrupt is generated. When the first interrupt is detected, the serial line is checked to ensure that it is in a spacing condition (valid START bit). If it is, the timer is set to P (to sample the middle of the first data bit) and a return is made to the program which was running when the

All mnemonics copyrighted © Intel Corporation 1976.

interrupt occurred. If the serial line has returned to the MARK state, a status flag is set to indicate an error and a return is made. On subsequent interrupt detection, the data is sampled, the timer is reinitiated, and control is returned to the program which was running when the interrupt occurred. When the last (i.e. STOP) bit is detected a completion flag is set and a return is made to the program running when the timer overflow occurred. By periodically checking the error and completion flags the running program can determine when the interrupt driven receive program has a character assembled for it.

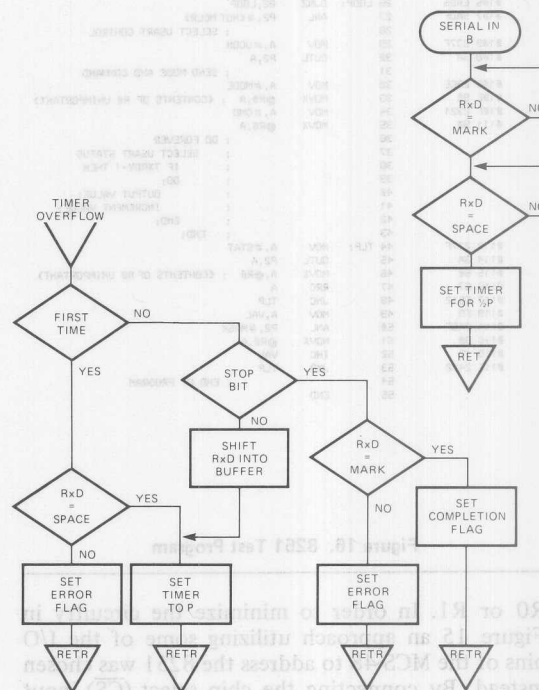


Figure 19. Improved Serial Input Routine

Using the timer to implement time delays as shown in Figure 19 results in considerable savings in processing time; two problems remain, however, which must be solved before an adequate software solution to the problem of receiving serial code can be found. The first problem is that even though the delays between bit samples are implemented via the timer rather than program loops the loop construction is still used to detect the leading edge of

the START bit. Although this results in the waste of processing power, the second problem is even more serious. For longer messages the required accuracy of the clocks becomes more and more stringent. Using the sampling technique discussed a cumulative error of one half a bit time in the time at which a bit sample is taken will result in erroneous reception. The maximum timing error which can be tolerated and yet still allow proper detection of an 11 bit ASCII character is then:

$$E_{max} = \frac{0.5 \cdot \text{BIT TIME}}{\text{CHARACTER TIME}} - \frac{0.5P}{11P} = 4.5\%$$

where P is the period of single bit. The corresponding calculation for a 32 bit character yields:

$$E_{max} = \frac{0.5P}{32P} = 1.6\%$$

Since this calculation does not allow for distortion on the signals, it is obvious that either extremely stable clocks will be required or a more tolerant algorithm must be devised. This problem is particularly serious at relatively high baud rates where the resolution of the counter (80μsecs with a 6 MHz crystal) becomes a significant percentage of the period of the received signal. At the 110 baud rate of the Teletype the 80μsec resolution of the clock allows a maximum accuracy of 0.33%; at 2400 baud this figure is reduced to 3.8%.

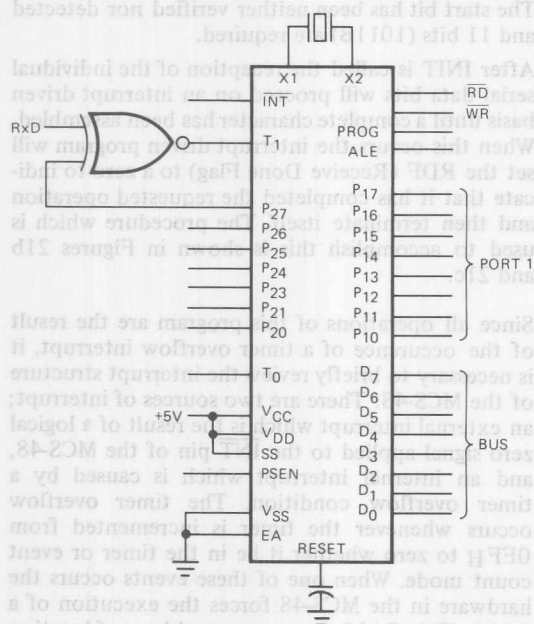


Figure 20. Detecting RxD Edges

Both efficient detection of the start bit and increased timing accuracy can be obtained if the MCS-48 can detect edges on the incoming received data (RxD). A hardware construct which allows this is shown in Figure 20.

The received data (RxD) is Exclusive NORed with bit seven of port two and fed into the TEST (T1) pin of the MCS-48. By manipulating P27 the program can now cause T1 to be either RxD or $\overline{\text{RxD}}$. (If P27 = 1 then T1 = RxD; if P27 = 0 then T1 = $\overline{\text{RxD}}$.) Note that not only can T1 be tested directly by the software but that it is the input which is used when the MCS-48 timer is in the event counter mode. The significance of this will be discussed later. The relationship between T1, P27, and RxD is given by the Boolean expression:

$$\overline{\text{T1}} = \text{P27} \cdot \overline{\text{RxD}} + \overline{\text{P27}} \cdot \text{RxD}$$

Figure 21 flowcharts a means of utilizing this hardware construct to avoid the necessity of wasting time in program loops to detect the leading edge of the start bit. The receive operation is initialized when the program desiring to receive serial data calls the INIT subroutine (Figure 21a). Since INIT is going to manipulate the timer the first action it performs is to disable the timer overflow interrupt. Its next step is to set P27 to a logical 1. Setting P27 in this manner causes the TEST 1 input to the MCS-48 to follow $\overline{\text{RxD}}$. By setting up the receive circuitry in this manner a high to low transition will occur on TEST 1 when the RxD goes from the MARKING to SPACING state (i.e. the START

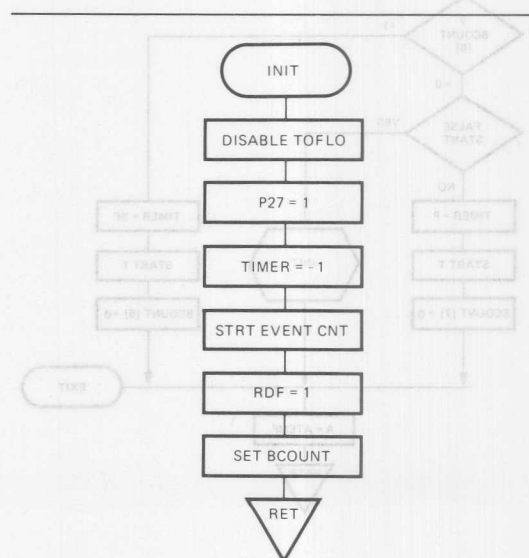


Figure 21a. Interrupt Driven Serial Receive Flowchart

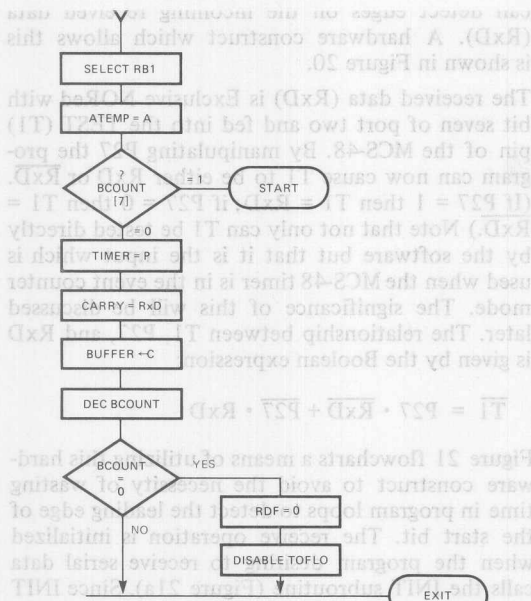


Figure 21b. Interrupt Driven Serial Receive Flowchart

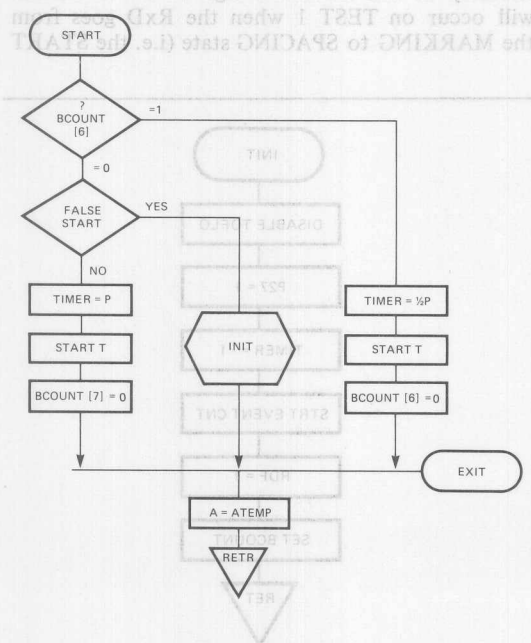
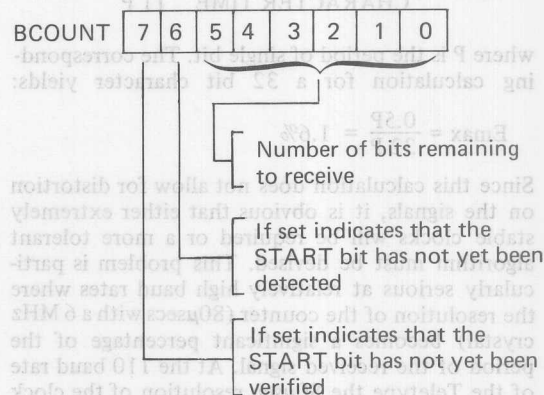


Figure 21c. Interrupt Driven Serial Receive Flowchart

overflow interrupt on the next MARK to SPACE transition of RxD (the TEST 1 input doubles as the event counter input). Before returning to the calling program the INIT routine sets a flag (RDF) which will be cleared by the receive program when the requested receive operation is complete. INIT also sets a value into a register called BCOUNT. The receive program interprets BCOUNT as follows:

Number of bits remaining
to receive

If set indicates that the START bit has not yet been detected

If set indicates that the START bit has not yet been verified

In order to request the reception of the 11 bit ASCII code INIT would set BCOUNT to 1100101B. The start bit has been neither verified nor detected and 11 bits (1011B) are required.

After INIT is called the reception of the individual serial data bits will proceed on an interrupt driven basis until a complete character has been assembled. When this occurs the interrupt driven program will set the RDF (Receive Done Flag) to a zero to indicate that it has completed the requested operation and then terminate itself. The procedure which is used to accomplish this is shown in Figures 21b and 21c.

Since all operations of this program are the result of the occurrence of a timer overflow interrupt, it is necessary to briefly review the interrupt structure of the MCS-48. There are two sources of interrupt; an external interrupt which is the result of a logical zero signal applied to the $\overline{\text{INT}}$ pin of the MCS-48, and an internal interrupt which is caused by a timer overflow condition. The timer overflow occurs whenever the timer is incremented from OFFH to zero whether it be in the timer or event count mode. When one of these events occurs the hardware in the MCS-48 forces the execution of a CALL. This CALL has a preset address of location 3 if it is due to the external interrupt and location 7 if it is due to a timer overflow. If both of these

events occur simultaneously the external interrupt will take precedence. The CALL automatically saves the contents of the program counter for the running program and its PSW (program status word) on a stack the hardware maintains in RAM locations 8-23. Although the hardware saves the program counter and PSW, it remains the responsibility of any interrupt driven software to make absolutely certain that it does not modify any memory locations or registers which are being used by the main program. The most convenient way of ensuring this in the MCS-48 is to dedicate the second bank of registers (RB1) to the interrupt driven program. One of these registers has to be used to save the accumulator (which is not part of the register bank) but seven registers remain; including two which can be used as pointers to the rest of the RAM (R0 and R1). Note that if this approach is taken then these registers have to be allocated between the program which services the external interrupt and the one which services the timer overflow. This problem is somewhat alleviated by a hardware lockout which prevents the timer overflow interrupt from interrupting the external interrupt service routine and vice versa. This is implemented by locking out new interrupts between the time an interrupt is recognized and the time a RETR instruction is executed. The RETR instruction is like a normal RET (return from subroutine) except that the PSW as well as the program counter is restored. The RETR instruction can be very much thought of as a return from interrupt instruction in the MCS-48.

The receive program under discussion uses register bank 1 in the manner described. Whenever a timer overflow occurs (e.g. on the next MARK to SPACE transition of Rx/D after INIT is called), control is passed (by the hardware generated CALL) to the point labeled TIMER OFLO in Figure 21b. This program segment immediately selects register bank 1 (RB1) and then saves the accumulator (A) in a location called ATEMP which is actually R7 of RB1. The program then tests bit seven of BCOUNT (R6 of RB1) to find out if a START bit has been verified (i.e. the edge of the START bit has first been detected and then verified to still be a SPACE one-half a bit time later. If BCOUNT [7] is a zero the START has been verified and the program proceeds to set the timer to P (the period of the serial bit), get the current serial data into the carry bit, and then shift the carry bit into a buffer. After saving the data the program decrements BCOUNT and tests it for zero. If BCOUNT is zero the receive operation is complete so the program sets RDF to a zero and disables timer overflow interrupts. Whether or not BCOUNT is zero, control is passed to EXIT where A is loaded with ATEMP and a

RETR is executed. Note that since the state of the flip flop which selects RB1 is saved as part of the PSW, the execution of RETR automatically selects the register bank which was active when the interrupt occurred.

If BCOUNT [7] is still set when it is tested, control is passed to START (Figure 21c) where bit 6 is tested to determine if the START has been detected yet. If BCOUNT [6] is set it indicates that this is the first occurrence of a timer overflow since the receive process was initialized by the INIT subroutine. If this is so, the program assumes that the START bit has just started and therefore it sets the timer to one-half of a bit time ($1/2 P$), starts the timer in the timer mode, and clears BCOUNT [6] to indicate that the START bit has been detected. The next overflow will again result in the execution of the program in Figure 21b and again BCOUNT [7] will be found to be set. This time, however, BCOUNT [6] will be reset and the program will know that it should test the START bit to ensure that it is still a SPACE. This test is performed and if successful the timer is set for a bit period P and BCOUNT [7] is reset so that on the next occurrence of a timer overflow the program will know that it should start assembling serial bits into a character. If the test is unsuccessful, the subroutine INIT is used to reinitialize the receive program. In either case control is passed to EXIT where a return from interrupt mode occurs.

This receive program, listings of which appear in Figure 22, allows the reception of serial characters transparently to the main running software. After INIT is called the main program has only to check RDF periodically to find out if there is data in the buffer for it. It would be fairly easy to 'double buffer' this operation by providing a buffer which the receive program uses to deserialize the incoming code and a second buffer to store the assembled character. If the program would reinitialize itself upon completion, the reception of a string of characters could proceed in much the same way as it would if a status driven USART were being used.

Although this program solves the first problem of software controlled reception (lack of efficiency) the second problem—sensitivity to frequency variations—remains. An example of a code which would be susceptible to this problem is the 31,26 BCH code commonly used in supervisory control systems. (A supervisory control system is, in essence, a remote control system which allows a human or computer operator the control of a system via a serial communications link.) The BCH codes are used because of their error detection capabilities and are a class of cyclical redundancy

RETR is executed. Note that since the state of the flip flop which selects RBT is saved as part of the P2W, the execution of RETR automatically selects the register bank which was active when the interrupt occurred.

```

LOC  OBJ      SEQ      SOURCE STATEMENT
1:      0
2:      1: *****
3:      2: SERIAL INPUT USING THE MCS-48
4:      3: THIS CODE ASSUMES HARDWARE
5:      4: SHOW IN P20.28. TO USE
6:      5: THIS ROUTINE CALL INIT.
7:      6: WHEN RDT=0 THE ASSEMBLED
8:      7: CHARACTER WILL BE IN SERBUF
9:      8:
10:     9: *****
11:     10:
12:     11:
13:     12: EQUATES
14:     13:
15:     14:
16:     15:
17:     16: ATMP EQU R7 ; STORAGE FOR A DURING INTERRUPT
18:     17: BCOUNT EQU R6 ; CONTAINS NUMBER OF BITS IN MSG
19:     18: COUNT EQU R2 ; UTILITY COUNTER
20:     19: RX8 EQU R8 ; POINTER
21:     20: BITNO EQU 0 ; NUMBER OF BITS
22:     21: P EQU 41 ; SAMPLE PERIOD
23:     22: SERBUF EQU 28H ; SERIAL BUFFER
24:     23: RDT EQU 24H ; RECEIVE DONE FLAG
25:     24:
26:     25:
27:     26: CONTROL PASSED HERE WHEN TIMER OFLO OCCURS
28:     27:
29:     28:
30:     29: ORG 07H ; *ENTER INTERRUPT MODE*/
31:     30: INVEC: SEL RBT1 ;
32:     31: MOV ATMP,A ;
33:     32: ; IF BCOUNT(7)=0 THEN
34:     33: MOV A,BCOUNT
35:     34: JNB START ; DO:
36:     35: ; TIMER=P;
37:     36:
38:     37: MOV A,#P
39:     38: MOV T,A ; START TIMER
40:     39:
41:     40: SLB: STRT T ;
42:     41:
43:     42: ; *CARRY-RDT/
44:     43: IN A,P2 ; CARRY=P27 AND TEST1;
45:     44: RLC A ;
46:     45: TISRDI ;
47:     46: CPL C ;
48:     47: ; *SHIFT CARRY INTO BUFFER*
49:     48:
50:     49:
51:     50: TISRDI: MOV RX8,#SERBUF
52:     51: SLDOP: XCH A,@RX8
53:     52: RRC A ;
54:     53: XCH A,@RX8
55:     54: ; BCOUNT=BCOUNT-1;
56:     55: ; IF BCOUNT=0 THEN
57:     56: DJNZ BCOUNT,SEXT
58:     57: DO:
59:     58: ; RDT=0;
60:     59: ; DISABLE EX INT;
61:     60: ;
62:     61: MOV RX8,#RDT
63:     62: CLR A ;
64:     63: @RX8,A ;
65:     64: DIS TCNT1
66:     65: ; END;
67:     66: JNP SEXT
68:     67: ; ELSE
69:     68: ; DO:
70:     69: ; IF BCOUNT(6)=0 THEN

```

Figure 22. Interrupt Driven Serial Receive Program

codes such as those used in synchronous data communications (e.g. BISYNC or SDLC). BCH codes, named for their originators Bose, Chaudhuri, and Hocquenghem, are characterized by having a length of $n=2^m-1$. The number of redundant check bits can be mt where t is a positive integer (clearly $mt \leq n$). The 31,26 code fits this format with $m=5$ and $t=1$. The length of each message is $n=2^5-1=31$ with 5*1 redundant bits, leaving 26 bits available for data transmission. With an appropriate poly-

nominal BCH codes can detect all errors consisting of 2t error bits and all burst errors of mt or fewer bits. The 31,26 BCH code will therefore detect any erroneous messages with 1 or 2 errors or bursts of errors of less than 5 bits. The 31,26 format (shown in Figure 23) requires the reception of a start bit followed by 31 information bits, clearly beyond the capability of the USART but perhaps within reach of a program controlled approach using the MCS-48 itself.

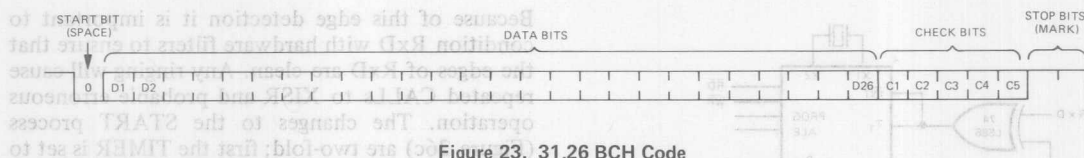


Figure 23. 31,26 BCH Code

A concept which reduces sensitivity to frequency deviations and thus allows the reception of longer codes is shown pictorially in Figure 24. The first line of this timing chart shows an alternative ones and zeros pattern on the RxD with a period of 5 milliseconds. The second line shows that by sampling at a period of exactly 5 milliseconds the data can be properly interpreted. The third and fourth lines show the effects of sampling with a period of six and four milliseconds respectively. In either case, an error occurs at the third sample where both periods result in sampling on an edge of the RxD signal. The third line of Figure 24 shows a hybrid sampling scheme which, based on some additional information, switches sampling periods between the two values. As can be seen in Figure 24, the data is sampled with a 4 millisecond period until the sampling begins to fall behind the data; at this point the sampling period is increased to six milliseconds and the sampling first catches up and then passes the center point of the data. As soon as this happens, the sampling period reverts to the 4 millisecond period and the cycle repeats. It can be seen that this scheme sets up a pattern which repeats indefinitely and the data can be successfully sampled. Note that the sampling pattern established is alternating periods of four and six milliseconds. The average period of this pattern, as might be expected, is 5 msec. Line 5 of Figure 24 shows the effect of a change in transmission speed to a period of 5.5 msec with no change in the sampling time. The sampling is again successful but the new sampling pattern is 4-6-6-6; 4-6-6-6, etc. Note that the average sample is again equal to the period of the received data (5.5). While this scheme

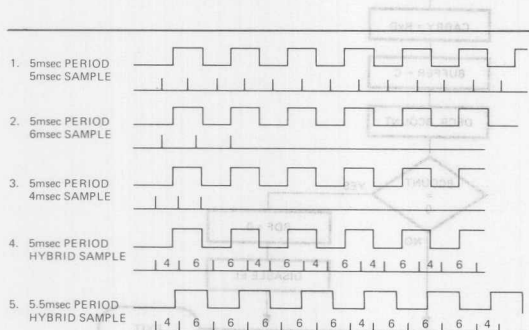


Figure 24. Various Sampling Alternatives

does seem to work, the question of what additional information is needed remains.

The MCS-48 must somehow decide when it is drifting out of synchronization and take corrective action. By referring back to Figure 24 it can be seen that if the MCS-48 could determine where the edges of RxD occurred with respect to its sampling times then the additional information would be available. As can be seen in the figure the choice of sampling period can be based on the following rule:

If an edge on the RxD line occurs during the first half of the current sampling period, then use the short period for the next sample. If an edge occurs during the second half of the period, then use the long sampling period for the next sample.

If the data on the RxD line does not change, of course, the MCS-48 will drift out of synchronization just as the original algorithm did. As long as edges occur on TxD, however, synchronization can be maintained. To maximize the allowable time between edges, the following addition could be made to the above rule:

If no edge occurs on the RxD line during a sample, then change sampling period from short to long or vice versa.

Note that this addition to the rule will result in using an average of the two sampling periods when no edge occurs for several bit times.

The edges of RxD can be easily detected by the use of the same structure (the Exclusive - NOR gate) which was added to the MCS-48 in Figure 20. This gate, which is used to detect the edge on RxD which begins the START bit, can naturally be used to detect any edge. Since the timer is being used to time the bit period, however, the event count input (T1) is not useful during the receive itself. By connecting the output of this gate, however, to the INT input to the MCS-48 (see Figure 25) it is possible to detect edges on RxD with the event counter when the program is trying to detect the START bit and by the external interrupt when the program is using the timer to control the sampling times.

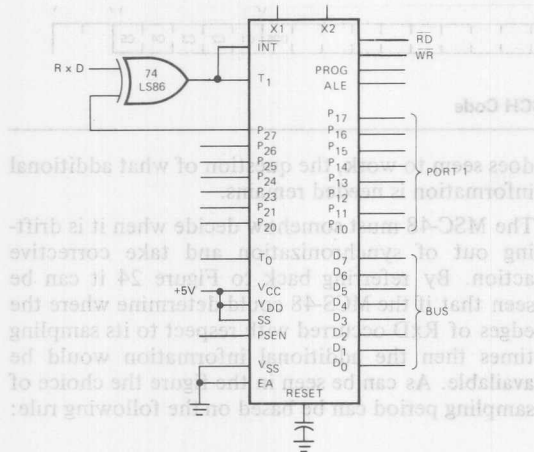
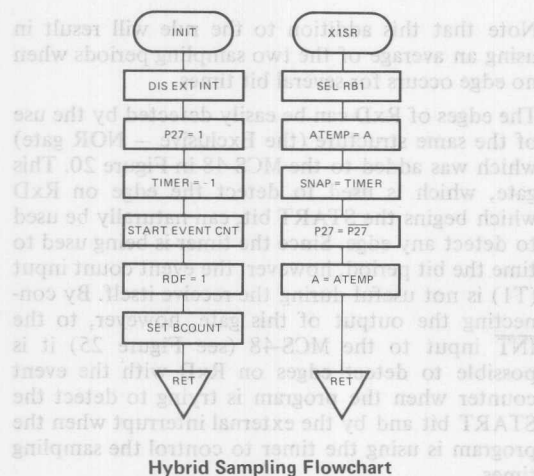


Figure 25. Modified Edge Detection

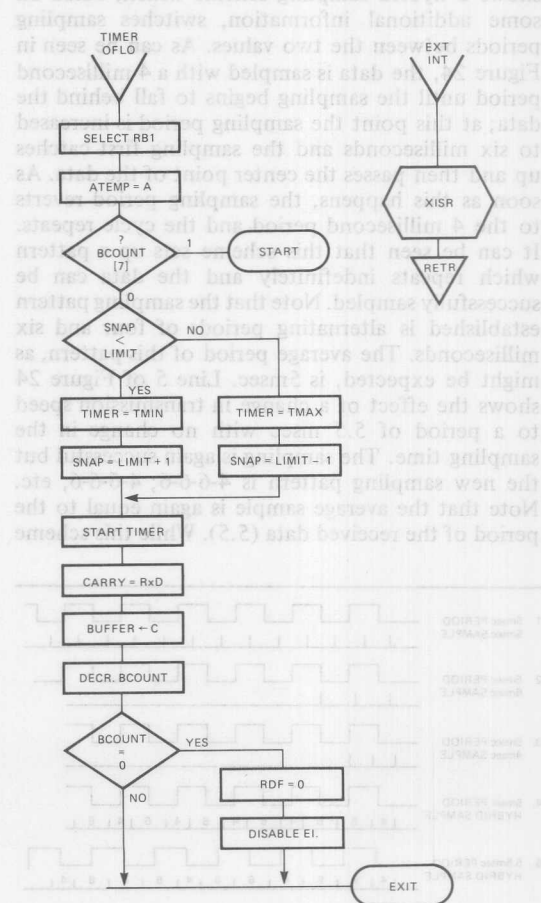
A modification to the program of Figure 21 which implements this new sampling algorithm is shown in Figure 26. The first deviation from the original program is the addition of a routine (XISR, Figure 26a which is called when an external interrupt occurs (i.e. when an edge occurs on RxD). This routine saves the status of the running program and then stores the current value of the timer register in a location called SNAP (R5 of RB1). After doing these operations the program complements bit 7 of port 2. Manipulating P27 in this manner will cause the Exclusive NOR gate to turn off the external interrupt and will set it up to generate another interrupt when the RxD line changes again (has another edge).



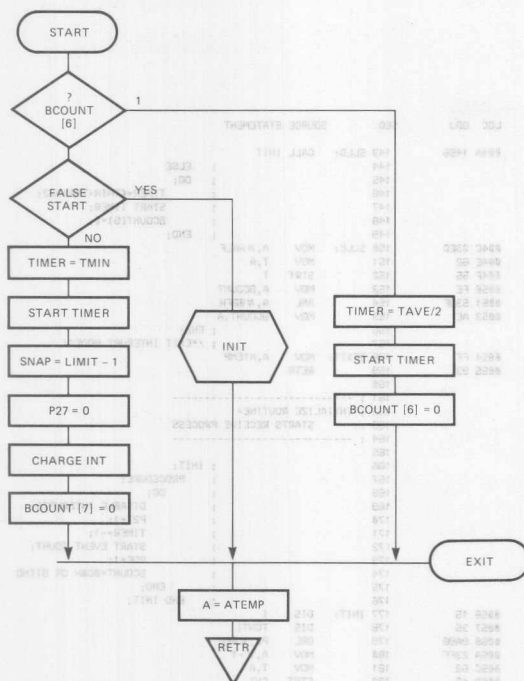
Hybrid Sampling Flowchart

the edges of RxD are clean. Any ringing will cause repeated CALLs to XISR and probable erroneous operation. The changes to the START process (Figure 26c) are two-fold; first the TIMER is set to one half the average of the two sample periods when the START bit is first detected (BCOUNT [6] = 1), and second the processing of the edge information is initialized by presetting SNAP and clearing P27.

SNAP is preset so that when the reception of data actually begins (Figure 26b BCOUNT [7] = 0), the decision block which tests SNAP against LIMIT will be initialized. This block actually compares the value in SNAP with a LIMIT value which is used to determine if the sampling point is ahead or behind the actual midpoint of the serial data. If the sampling is ahead then the timer is set for TMIN; if the sampling is behind then the timer is set for



Hybrid Sampling Flowchart



Hybrid Sampling Flowchart

TMAX. By presetting SNAP in the manner shown in the flowcharts the second rule of the algorithm, (if no edge appears on the Rx/D line during a sample, then change the sampling periods short to long or vice versa) is automatically met. If an edge occurs then XISR will modify SNAP, if XISR is not invoked between two samples then the choice of timer periods will alternate. The only other significant change to the algorithm is that the INIT routine must now lock-out all interrupts, not just the timer overflow interrupt, while it is operating. A program which uses this algorithm to receive a 32 bit message is shown in Figure 27.

LOC	OBJ	SEQ	SOURCE STATEMENT
0		1	*****
1		2	SERIAL INPUT USING MCS-48
2		3	THIS CODE ASSUMES HARDWARE
3		4	SHOWN IN FIG 25. PROGRAM
4		5	IS SIMILAR TO PREVIOUS
5		6	ONE. A MORE SOPHISTICATED
6		7	SAMPLING ALGORITHM IS USED
7		8	NOTE: A PL/M LIKE LANGUAGE WAS USED
8		9	TO COMMENT THIS LISTING AND
9		10	SEVERAL OTHERS IN THIS NOTE. NO
10		11	COMPILER EXISTS FOR THE MCS-48.
11		12	THE COMMENTS WERE *HAND
12		13	COMPILED* INTO ASSEMBLY CODE
13		14	*****
14		15	*****
15		16	*****
16		17	*****
17		18	*****
18		19	*****
19		20	EQUATES
20		21	*****
21		22	*****
22		23	ATEMP EQU R7 ; STORAGE FOR A DURING INTERRUPT
23		24	BCOUNT EQU R6 ; CONTAINS NUMBER OF BITS IN MSG
24		25	SNAP EQU R5 ; TAKES TIMER SNAP SHOT ON RXD EDGE
25		26	COUNT EQU R2 ; UTILITY COUNTER
26		27	RXD EQU R0 ; POINTER
27		28	BITNO EQU 32 ; NUMBER OF BITS
28		29	LIMIT EQU 20 ; TEST VALUE FOR MIN/MAX SAMPLING
29		30	TMAX EQU 43 ; MAX SAMPLE PERIOD
30		31	TMIN EQU 39 ; MINIMUM SAMPLE PERIOD
31		32	HALF EQU 20 ; HALF NOMINAL PERIOD
32		33	SERBUF EQU 20H ; START OF SERIAL BUFFER
33		34	RDF EQU 24H ; RECEIVE DONE FLAG
34		35	*****
35		36	*****
36		37	CONTROL PASSED HERE ON EXT. INT.
37		38	*****
38		39	*****
39		40	ORG 03H
40		41	CALL XISR ; CALL SERVICE ROUTINE
41		42	EIVC: RETR
42		43	*****
43		44	*****
44		45	*****
45		46	CONTROL PASSED HERE WHEN TIMER OFLO OCCURS
46		47	*****
47		48	*****
48		49	*****
49		50	*****
50		51	*****
51		52	*****
52		53	*****
53		54	*****
54		55	*****
55		56	*****
56		57	*****
57		58	*****
58		59	*****
59		60	*****
60		61	*****
61		62	*****
62		63	*****
63		64	*****
64		65	*****
65		66	*****
66		67	*****
67		68	*****
68		69	*****
69		70	*****
70		71	*****
71		72	*****
72		73	*****
73		74	*****

Figure 27. Hybrid Sampling Program

LOC	OBJ	SEQ	SOURCE STATEMENT
0019	62	74	MOV T,A
001A	BD13	75	MOV SNAP,#LIMIT-1
		76	START TIMER;
001C	55	77	SLLB: STRT T
		78	/*CARRY=RXD*/
		79	CARRY=P27 XOR TEST1;
001D	0A	80	IN A,P2
001E	F7	81	RLC A
001F	4622	82	JMT1 TISRD
0021	A7	83	CPL C
		84	/*SHIFT CARRY INTO BUFFER*/
		85	;
		86	;
		87	;
		88	;
		89	;
		90	;
		91	;
		92	TISRD: MOV RX0,#SERBUF
0022	B820	93	MOV COUNT,#4
0024	BA04	94	SLOOP XCH A,RX0
0026	20	95	RRC A
0027	67	96	XCH A,@RX0
0028	20	97	INC RX0
0029	10	98	DJNZ COUNT,SLOOP
002A	EA26	99	;
		100	;
		101	;
002C	EE54	102	DJNZ BCOUNT,SEXIT
		103	DO;
		104	;
		105	;
		106	;
002E	B824	107	CLR A
0030	27	108	MOV @RX0,A
0031	A0	109	DIS TONTI
0032	25	110	DIS I
0033	15	111	END;
0034	0454	112	JMP EXIT
		113	;
		114	;
		115	;
		116	;
0036	FE	117	START: MOV A,BCOUNT
0037	D24C	118	JNB SLLC
		119	DO;
		120	;
0039	564A	121	JT1 SLLD
		122	DO;
		123	;
		124	;
		125	;
		126	;
		127	;
		128	;
003B	23D9	129	MOV A,#TMIN
003D	62	130	MOV T,A
003E	55	131	STRT T
003F	BD15	132	MOV SNAP,#LIMIT-1
0041	9A7F	133	ANL P2,#7FH
0043	05	134	EN I
0044	FE	135	MOV A,BCOUNT
0045	537F	136	ANL A,#7FH
0047	AE	137	MOV BCOUNT,A
0048	0454	138	JMP EXIT
		139	;
		140	;
		141	;
		142	;

LOC	OBJ	SEQ	SOURCE STATEMENT
004A	1456	143	SLLD: CALL INIT
		144	;
		145	;
		146	;
		147	;
		148	;
		149	;
004C	23EC	150	SLLC: MOV A,#HALF
004E	62	151	MOV T,A
004F	55	152	STRT T
0050	FE	153	MOV A,BCOUNT
0051	53BF	154	ANL A,#0BFH
0053	AE	155	MOV BCOUNT,A
		156	;
		157	;
0054	FF	158	SEXIT: MOV A,ATEMP
0055	93	159	RETR
		160	;
		161	;
		162	;
		163	;
		164	;
		165	;
		166	;
		167	;
		168	;
		169	;
		170	;
		171	;
		172	;
		173	;
		174	;
		175	;
		176	;
0056	15	177	INIT: DIS I
0057	35	178	DIS TONTI
0058	0AB6	179	ORL P2,#0BH
005A	23FF	180	MOV A,#-1
005C	62	181	MOV T,A
005D	45	182	STRT CNT
005E	B824	183	MOV RX0,#RDF
0060	F3	184	MOV A,B1
0061	A0	185	MOV @RX0,A
0062	25	186	EN TONTI
0063	BEE0	187	MOV BCOUNT,#0B0H OR BITNO
0065	03	188	RET
		189	;
		190	;
		191	;
		192	;
		193	;
		194	;
		195	;
		196	;
		197	;
		198	;
		199	;
		200	;
0066	D5	201	XISR: SEL RB1
0067	AF	202	MOV ATEMP,A
0069	42	203	MOV A,T
0069	AD	204	MOV SNAP,A
006A	0A	205	IN A,P2
006B	D300	206	XRL A,#0BH
006D	3A	207	OUTL P2,A
006E	FF	208	MOV A,ATEMP
006F	03	209	RET
		210	;
		211	END

Figure 27. Hybrid Sampling Program

TRANSMITTING SERIAL CODE

Serial transmission is conceptually far simpler than serial reception since no synchronization is required. All that is required is to use the timer to generate interrupts at the bit rate and present the character to be transmitted serially at an I/O pin. A program which does this is shown in Figure 28. The transmission of serial data becomes much more complicated if it must occur simultaneously with reception.

If both reception and transmission are to occur simultaneously then obviously contention will exist for the use of the timer. It is possible to allow the simultaneous reception and transmission of serial data using the timer as a general clock which controls software maintained timers. The attainable baud rates using such techniques are, however, limited and the use of a 8251 USART is probably

indicated in all but the most cost sensitive applications. An exception to this rule occurs when the system, although full duplex in nature, actually transmits the same data as it receives. An example of this is a microprocessor driving a terminal such as a Teletype. Although the circuit to the terminal is full duplex, the data that is transmitted is generally the same as that received. A minor modification to the program shown in Figure 26 would implement this mode of operation. The modification would be to the XISR routine and it would add the code necessary to place the TxD I/O pin in the same state as the RxD line. Since any change in RxD results in a call to XISR, this modification would cause the retransmission of any received data. Whenever it becomes necessary to transmit data which is not being received, the program of Figure 28 could be used in a half duplex manner.

```

LOC  OBJ  SEQ  SOURCE STATEMENT
0
1  ; -----
2  ; SERIAL TRANSMIT ON THE MCS48
3  ; TO USE PUT A CHAR IN BUFF AND
4  ; SET CHARAV TO 0FFH. WHEN THE
5  ; TRANSMITTER IS READY FOR ANOTHER
6  ; CHAR IT WILL CLEAR CHARAV. THE
7  ; TRANSMISSION IS DOUBLE BUFFERED.
8  ; -----
9
10 ; -----
11 ; EQUATES
12 ; -----
13
0007 14 ATEMP EQU R7 ; STORAGE FOR A DURING INT.
0006 15 PTOS EQU R6 ; PARALLEL TO SERIAL CONVERTER
0005 16 BUFF EQU R5 ; CHARACTER BUFFER
0004 17 CHARAV EQU R4 ; CHARACTER AVAILABLE FLAG
0003 18 COUNT EQU R3 ; BIT COUNTER
000F 19 CBIT EQU 0EFH ; MASK TO CLEAR TXD IN P24
0010 20 SBIT EQU 010H ; MASK TO SET TXD IN P24
FFD7 21 P EQU -41 ; PERIOD OF TXD
22
23 ; -----
24 ; CONTROL PASSED HERE ON TIMER OVERFLOW
25 ; -----
0007 26 ORG 07H ; ENTER INTERRUPT MODE
0007 D5 28 TDFLD: SEL RB1
0008 AF 29 MOV ATEMP,A
30 ; SET TIMER FOR P
0009 23D7 31 MOV A,#P
000B 62 32 MOV T,A
000C 55 33 STRT T
34 ; GET BIT INTO CARRY
000D 141D 35 CALL BIT
36 ; SET TXD TO CARRY

```

```

LOC  OBJ  SEQ  SOURCE STATEMENT
000F 0A 37 IN A,P2
0010 D300 38 XRL A,#0BH
0012 2A 39 OUTL P2,A
0013 F619 40 JC BITON
0015 9AEF 41 ANL P2,#CBIT
0017 041B 42 JMP EXIT
0019 0A10 43 BITON: ORL P2,#SBIT
001B FF 44 EXIT: MOV A,ATEMP
001C 93 45 RETR
46
47 ; -----
48 ; BIT ROUTINE
49 ; -PICKS THE NEXT BIT TO TRANSMIT
50 ; -----
51
001D FB 52 BIT: MOV A,COUNT
001E C627 53 JZ IDLE
0020 FE 54 MOV A,PTOS
0021 67 55 RRC A
0022 4300 56 ORL A,#0BH
0024 AE 57 MOV ATEMP,A
0025 CB 58 DEC COUNT
0026 03 59 RET
60
0027 97 61 IDLE: CLR C
0028 FC 62 MOV A,CHARAV
0029 962D 63 JNZ GOTONE
002B A7 64 CPL C
002C B3 65 RET
66
002D FD 67 GOTONE: MOV A,BUFF
002E AE 68 MOV ATEMP,A
002F 0B0A 69 MOV COUNT,#10
0031 5C00 70 MOV CHARAV,#0
0033 03 71 RET
72 ; END OF PROGRAM
73 END

```

Figure 28. Serial Transmission

it can be programmed to automatically generate and check parity. If the communications is handled by software within the MCS-48™ then the program must perform parity calculations. Calculating parity is easy if one remembers what parity really means. A character has even parity if the number of one bits in it is even. A character has odd parity if it has an odd number of ones. The program segment shown in Figure 29 can be caused to calculate parity. It starts by setting a loop count to eight and

results in a call to XISR this modification would cause the transmission of any received data. Whenever it becomes necessary to transmit data which is not being received, the program of Figure 29 could be used in a half duplex manner.

```

LOG DBJ      SEQ      SOURCE STATEMENT
1
2 ; *****
3 ;
4 ; PARITY
5 ; THIS PROGRAM GENERATES PARITY
6 ; ON THE ACCUMULATOR
7 ; CARRY WILL BE SET IF A HAS ODD PARITY
8 ;
9 ; *****
10
11
12 ; -----
13 ; EQUATES
14 ; -----
15
16 COUNT EQU R2
17
18 PAR: ORG 18BH ; SET LOOP COUNT
19      MOV COUNT, #8 ; INITIALIZE CARRY
20      CLR C ; FOR EACH ZERO BIT IN A
21      TFR A, C ; COMPLEMENT THE CARRY FLAG
22
23 LOOP: RR A ; ROTATE
24      JNB C, OVER ; IF CARRY
25      CPL C ; OVER
26
27      ; END OF PROGRAM
28      END

```

Figure 29. Parity Generation

clearing the CARRY flag. After this initialization a loop is executed eight times. During each execution the accumulator is rotated and the least significant bit is tested. If the bit is a zero the CARRY flag is complemented, if the bit is a one no further action is taken. Since an even number of zeros implies an even number of ones for an eight bit character, after all eight loops have been accomplished the CARRY bit will be set if an odd number of ones were encountered; it will be reset if the number were even. Since the RR instruction does not involve CARRY the net result of executing this program loop is to set CARRY if parity is odd without effecting the character in the accumulator.

with the MCS-48™ family. The application of this new single chip computer system to tasks which have not yet yielded to the power of the micro-processor will present a fascinating challenge to the system designer.

If both reception and transmission are to occur simultaneously then obviously contention will exist for the use of the timer. It is possible to allow the simultaneous reception and transmission of serial data using the timer as a general clock which controls software maintained timers. The attainable baud rates using such techniques are, however, limited and the use of a 8251 USART is probably

LOG DBJ SEQ SOURCE STATEMENT
1
2 ; *****
3 ;
4 ; PARITY
5 ; THIS PROGRAM GENERATES PARITY
6 ; ON THE ACCUMULATOR
7 ; CARRY WILL BE SET IF A HAS ODD PARITY
8 ;
9 ; *****
10
11
12 ; -----
13 ; EQUATES
14 ; -----
15
16 COUNT EQU R2
17
18 PAR: ORG 18BH ; SET LOOP COUNT
19 MOV COUNT, #8 ; INITIALIZE CARRY
20 CLR C ; FOR EACH ZERO BIT IN A
21 TFR A, C ; COMPLEMENT THE CARRY FLAG
22
23 LOOP: RR A ; ROTATE
24 JNB C, OVER ; IF CARRY
25 CPL C ; OVER
26
27 ; END OF PROGRAM
28 END

ASSEMBLY CROSS-REFERENCE 3-56



INTRODUCTION

This application note presents a software package for interfacing members of Intel's MCS-48™ family of single-chip microcomputers with keyboards and displays using a minimum of external components. Because of the similarity of the architectures of the various members of the family (the 8035, 8048, 8748, 8039, 8049, 8021, and 8022 microcomputers; also the 8041 and 8741 universal peripheral interfaces in the UPI-41® family), the code included here could run with minor modifications on any member of the family.

Since keyboard and display logic can be just one of several functions handled by a microprocessor, the added cost of including these functions in a system is minimal. In fact, considering the extremely low cost of standard X-Y matrix keyboards and integrated displays, their use is often more cost effective than even a handful of discrete switches and indicators. Thus, the additional flexibility of keyboard input and display output can be added to inexpensive consumer products (such as games, clocks, thermostats, tape recorders, etc.), while producing a net savings in system cost.

Since each potential application will have its own unique combination of keys and display characters, the program is written so that very little modification is needed to interface it with a wide variety of hardware configurations. In general, the only changes required are within the set of initial EQUates at the beginning of the program.

Along with the basic software for driving a multiplexed display and/or scanning and debouncing an X-Y matrix of key switches, a collection of utility subroutines is also included for implementing the most commonly used keyboard and display utility functions, such as copying simple messages onto the display or determining the encoded value of each key in the key matrix. As a result of the versatile architecture and applications-oriented instruction set of the MCS-48 family, the entire package fits into about 250 bytes of internal program ROM or EPROM, leaving the rest of the ROM space for the program to cook the perfect piece of toast, or whatever. By tailoring the software to match a known hardware configuration, or by selecting only those functions needed for a given application, the program size could be even further reduced.

Since what is being presented in this application note is a software package, rather than the usual hardware/software system design, the format of this note is somewhat different from most — it consists primarily of a long program listing reproduced in the following pages. For the most part, the listing is self-explanatory, with comments introducing each subroutine and major code segment. Some parts of this introduction are reproduced in the program listing itself, explaining the configuration of the prototype system. However, an additional bit of explanation would make the listing easier to understand, especially for those readers unfamiliar with the concept of multiplexed displays and keyboards.

In traditional digital system design, various hardware registers or counters were used to hold binary or BCD values which had to be conveyed to the user. The standard way of presenting this information was by connecting each register to a seven-segment encoder (such as the 7447) driving a single display character, as represented by Figure 1. Thus, two ICs, seven current limiting resistors, and about 45 solder joints were required for each digit of output. Consider how traditional techniques might be (mis-)applied in designing a microprocessor system: the designer could add a latch, encoder, and resistors for each digit of the display. Still another latch and decoder could be used to turn on one of the decimal points (if used). The characters displayed could only be a sequence of decimal digits. In the same vein, a large matrix of key switches could be read by installing an MSI TTL priority encoder read by an additional input port. Not only would all this use a lot of extra I/O ports and increase the system price and part count drastically, but the flexibility and reliability of the system would be greatly reduced.

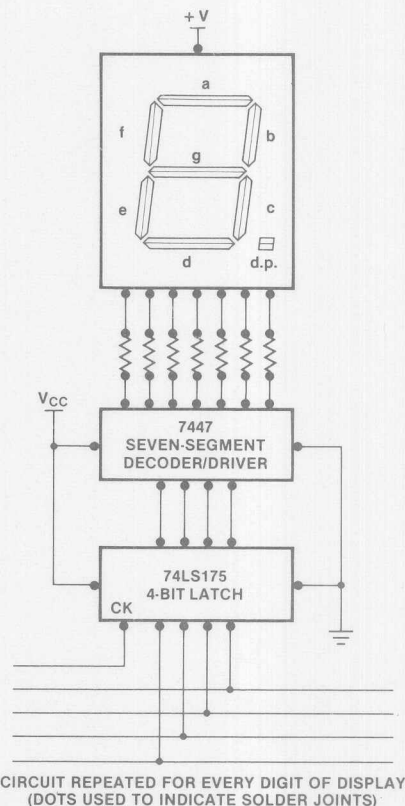


Figure 1. Wrong Way to Design Multiple Digit Displays for Microcomputer Systems

Instead, a scheme of time-multiplexing the display can be used to decrease costs, part count, and interconnections, while allowing a wider range of character types to be used on the display. The techniques used here are fairly typical of today's integrated subsystems designed especially for controlling keyboards and displays (such as in calculators or the Intel® 4269, 8278, and 8279 Keyboard/Display Controller Devices).

In a multiplexed display, all the segments of all the characters are interconnected in a regular two-dimensional array. One terminal of each segment is in common with the other segments of the same character; the other terminal is connected with the same segments of the other characters. This is represented schematically in Figure 2. A digit driver or segment driver is needed for each of these common lines.

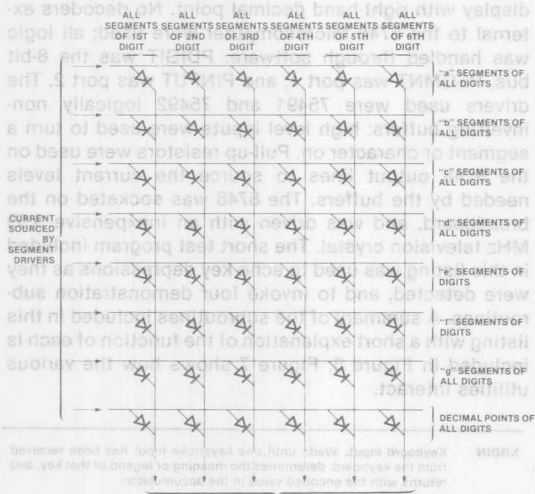


Figure 2. Schematic Representation of 6-Digit, 7-Segment Common-Cathode LED Multiplexed Display

The various characters of the display are not all on at once; rather, only one character at a time is energized. As each character is enabled, some combination of segment drivers is turned on, with the result that a digit appears on the enabled character. (For example, in Figure 3, if segment drivers 'a', 'b', and 'c' were on when character position #6 was enabled, the digit '7' would appear in the left-most place.) Each character is enabled in this way, in sequence, at a rate fast enough to ensure that the display characters seem to be on constantly, with no appearance of flashing or flickering.

In the system presented here, these rapid modifications to the display are all made under the control of the MCS-48™ microcomputer. At periodic intervals the computer quickly turns off all display segments, disables the character now being displayed and enables the next, looks up the pattern of segments for the next character

to be displayed, and turns on the appropriate segments. With the next character now turned on, the processor may now resume whatever it had been doing before. The whole display updating task consumes only a small fraction of the processor's time.

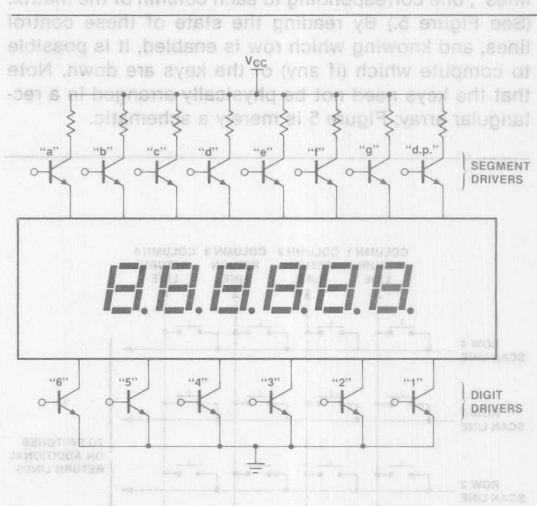


Figure 3. Segment and Digit Drivers used with 6-Position, 7-Segment LED Display

Moreover, since the computer rather than a standard decoder circuit is used to turn the segments off and on, patterns for characters other than decimal digits may be included in the display. Hexadecimal characters, special symbols, and many letters of the alphabet are possible. With sufficient imagination this feature can be exploited for some applications, as suggested by the examples in Figure 4.

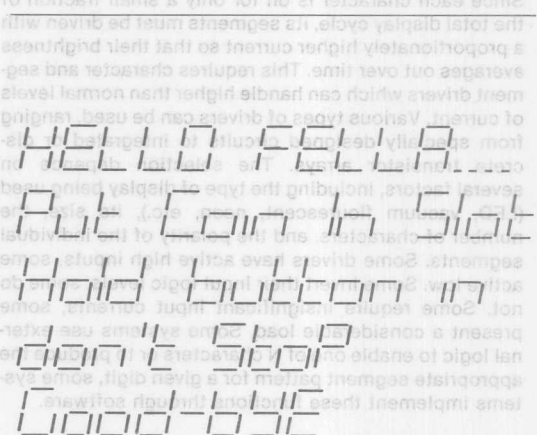


Figure 4. Examples of Typical Messages Possible with Simple 7-Segment Displays

will then pass the signal on to one of several "return lines", one corresponding to each column of the matrix. (See Figure 5.) By reading the state of these control lines, and knowing which row is enabled, it is possible to compute which (if any) of the keys are down. Note that the keys need not be physically arranged in a rectangular array; Figure 5 is merely a schematic.

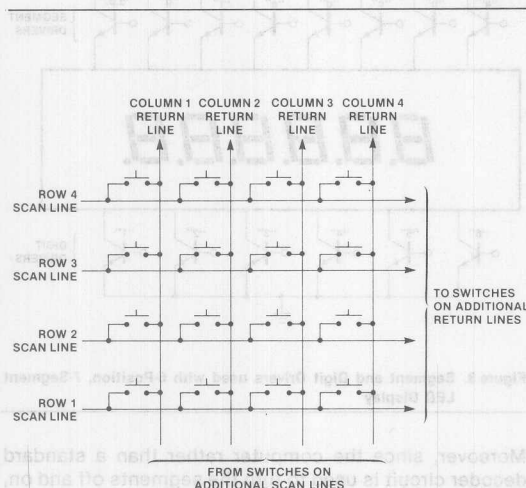


Figure 5. Schematic of X-Y Matrix Multiplexed Keyboard

Since each character is on for only a small fraction of the total display cycle, its segments must be driven with a proportionately higher current so that their brightness averages out over time. This requires character and segment drivers which can handle higher than normal levels of current. Various types of drivers can be used, ranging from specially designed circuits to integrated or discrete transistor arrays. The selection depends on several factors, including the type of display being used (LED, vacuum fluorescent, neon, etc.), its size, the number of characters, and the polarity of the individual segments. Some drivers have active high inputs, some active low. Some invert their input logic levels, some do not. Some require insignificant input currents, some present a considerable load. Some systems use external logic to enable one of N characters or to produce the appropriate segment pattern for a given digit, some systems implement these functions through software.

Because of these and the other variables which make each application unique, provisions are made in the first page of symbol EQUates to allow the user to specify such things as the number of characters in the display or the polarity of the drivers used, and the program will be assembled accordingly. The display is refreshed on each timer interrupt, which occurs every $32 \times (\text{TICK})$

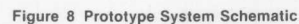
detained explanation of these variables is included in the listing.

Port assignment is also at the discretion of the user — all port references in the listing are "logical" rather than physical port names. The port used to specify which character is enabled is referred to as "PDIGIT". The output segment pattern is written to "PSGMNT" and the keyboard return lines are read by "PINPUT". These logical port names may be assigned to whichever ports the user pleases.

By way of example, the breadboard used to develop and debug this software used a matrix of 16 single-pole pushbuttons and an 8-character common-cathode LED display with right-hand decimal point. No decoders external to the 8748 microcomputer were used; all logic was handled through software. PDIGIT was the 8-bit bus, PSGMNT was port 1, and PINPUT was port 2. The drivers used were 75491 and 75492 logically non-inverting buffers: high level inputs were used to turn a segment or character on. Pull-up resistors were used on the 8748 output lines to source the current levels needed by the buffers. The 8748 was socketed on the breadboard, and was driven with an inexpensive 3.59 MHz television crystal. The short test program included in this listing was used to echo key depressions as they were detected, and to invoke four demonstration sub-routines. A summary of the subroutines included in this listing with a short explanation of the function of each is included in Figure 6; Figure 7 shows how the various utilities interact.

KBDIN	Keyboard Input. Waits until one keystroke input has been received from the keyboard; determines the meaning or legend of that key, and returns with the encoded value in the accumulator.
CLEAR	Blank out the display.
ENCACC	Encode accumulator with bit pattern corresponding to the segment pattern needed by the display to represent that symbol or character. Uses the value of the accumulator when called to access a table containing the patterns for all legal input values.
WDISP	Write into Display. Writes the bit pattern in the accumulator into the next character position of the display. Maintains a character position counter so that repeated calls will automatically write characters into sequential positions.
RENTRY	Right-hand Entry. Stores the accumulator segment pattern in the display in the right-most character position. Shifts all other characters to the left one place.
PRINT	Print a string of arbitrary characters onto the display. Useful for prompting messages, warnings, etc. Uses a table of segment patterns in ROM, so that messages will not be restricted to numbers, letters, etc.
FILL	Fill the display with the character pattern in the accumulator. Useful for writing dashes, segment test patterns, etc., into all character positions.
ECHO	Wait for a key to be pressed by the operator and write that key onto the display. Used for providing feedback to the operator when entering numeric data, etc.
RDPADD	Adds or deletes a decimal point to the character at the right-hand side of the display, for entering floating point numbers.
HOLD	Called when a key is known to be down. Does not return until all keys have been released. Used for organ-type keyboards, or when some action should not be initiated until the key invoking that action has been released.
DELAY	Provides a crude real-time delay corresponding to the value of the accumulator when called. Can be used to cause display characters to blink, to momentarily flash information, to enable a buzzer, etc. Could also be used by the program when delays are needed, such as to slow down the computer reaction rate while playing a game against the human operator.

Figure 6. Utility Subroutine Definitions



ISIS-II MCS-48/UP1-41 MACRO ASSEMBLER, V2.0
 AF40: INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX

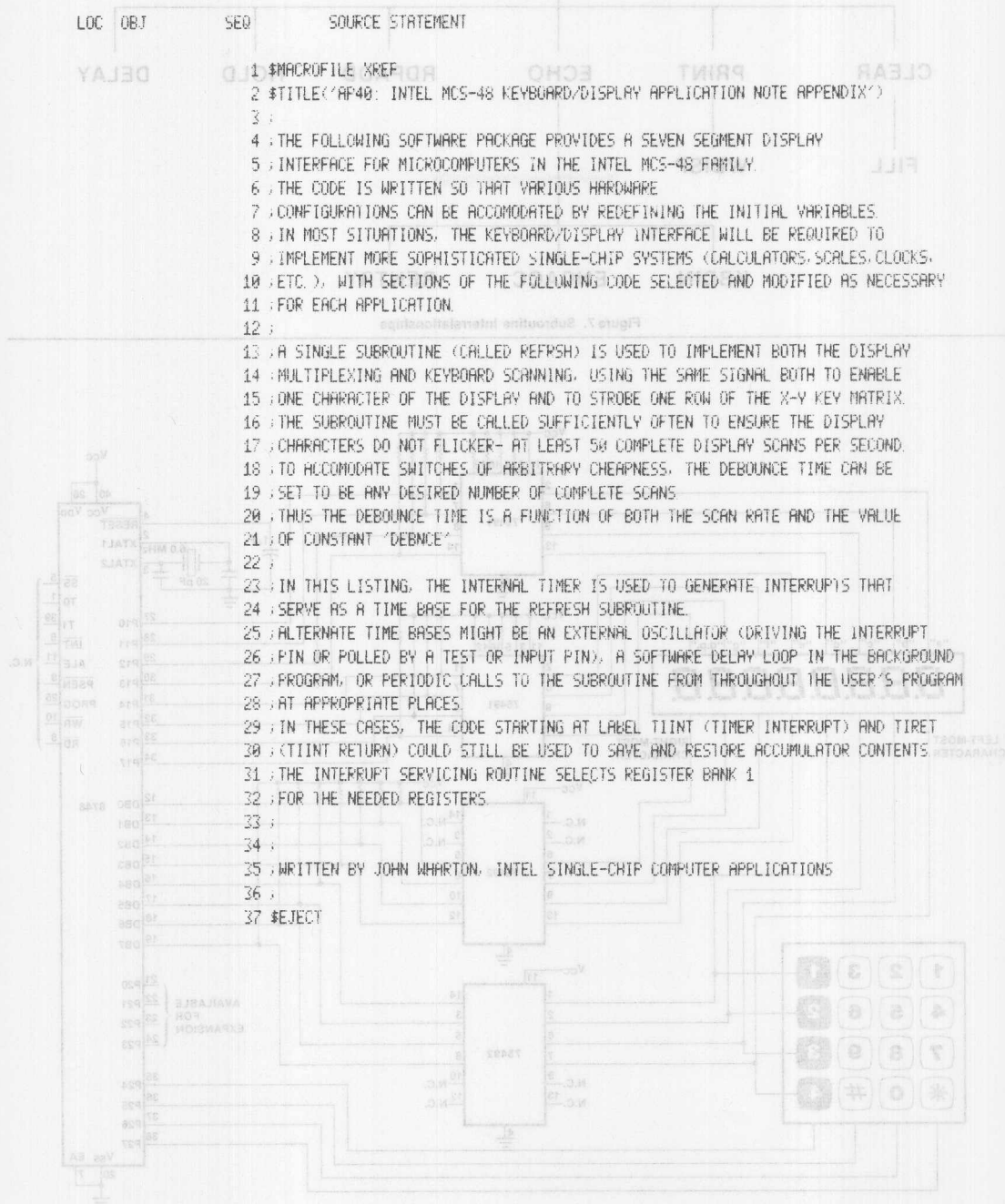


Figure 8 Prototype System Schematic

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

AP40: INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX

LOC	OBJ	SEQ	SOURCE STATEMENT
38			; IN THIS IMPLEMENTATION OF THE DISPLAY SCAN, IT IS ASSUMED THAT THERE WILL
39			; BE RELATIVELY LITTLE I/O OTHER THAN FOR THE KEYBOARD/DISPLAY.
40			; IF THIS IS THE CASE, THEN THERE IS NO NEED FOR FOR ANY ADDITIONAL EXTERNAL
41			; LOGIC (SUCH AS ONE-OF-EIGHT DECODERS OR SEVEN-SEGMENT ENCODERS), THOUGH
42			; THERE WILL STILL BE A NEED FOR CURRENT OR VOLTAGE DRIVERS, ACCORDING TO
43			; THE TYPE OF DISPLAY BEING USED.
44			;
45			; IN THIS LISTING, THE PROCESSOR I/O PORTS ARE LOGICALLY DIVIDED AS FOLLOWS:
46			;
47			; PDIGIT-EIGHT BIT PORT USED TO ENABLE, ONE AT A TIME, THE INDIVIDUAL
48			; CHARACTERS OF AN EIGHT DIGIT SEVEN-SEGMENT DISPLAY, WHILE ALSO
49			; STROBING THE ROWS OF AN X-Y MATRIX KEYBOARD.
50			; BIT7 ENABLES THE LEFTMOST CHARACTER AND THE BOTTOM ROW OF THE KBD,
51			; BIT4 ENABLES THE TOP ROW OF THE 4X4 KBD AND THE FOURTH CHARACTER,
52			; BIT0 ENABLES THE RIGHTMOST CHARACTER.
53			; (A 4X8 KEYBOARD COULD BE STROBED BY ALSO USING BIT3-BIT0
54			; AND EXTENDING OR ELIMINATING THE TABLE, "LEGND".)
55			; THE ENABLING OF ONE BIT (ACTIVE HIGH OR LOW) IS ACCOMMODATED BY
56			; ACCESSING A LOOK-UP TABLE CALLED CHRSTB.
57			; THIS TECHNIQUE TAKES ABOUT FOUR BYTES MORE ROM THAN A TECHNIQUE
58			; OF ROTATING A 'ONE' THROUGH A FIELD OF 'ZEROS' IN THE ACC
59			; AN APPROPRIATE NUMBER OF TIMES, BUT IT ALLOWS SOME ADDITIONAL
60			; FLEXIBILITY: IF THE DRIVERS BEING USED HAVE A COMBINATORIAL INPUT
61			; (AS IN THE 7545X FAMILY OF HIGH-CURRENT, HIGH-VOLTAGE DRIVERS),
62			; THE CHRSTB TABLE COULD PROVIDE ENCODED OUTPUTS, NINE DIGITS, FOR
63			; EXAMPLE, COULD BE ENABLED WITH SIX BITS OF (BUFFERED) OUTPUT:
64			; (001001, 001010, 001100, 010001, 010010, 010100, 100001, 100010, 100100)
65			; IF I/O LINES NEED TO BE CONSERVED, OR IF MANY DIGITS
66			; MUST BE DISPLAYED, AN EXTERNAL DECODER COULD BE ADDED TO THE SYSTEM.
67			; DURING CHARACTER TRANSITIONS A 'BLANK' CHARACTER IS
68			; EXPLICITLY WRITTEN TO THE DISPLAY. THUS,
69			; THERE WILL BE NO CHARACTER 'SHADOWING' CAUSED BY THE
70			; FACT THAT THE HARDWARE OR SOFTWARE DECODER KEEPS ONE
71			; OUTPUT, AND THUS ONE CHARACTER, ACTIVE AT ALL TIMES.
72			;
73			; PSGMNT-EIGHT BIT PORT TO ENABLE THE SEVEN SEGMENTS & D.P. OF A STANDARD
74			; DISPLAY.
75			; BIT7-BIT0 CORRESPOND TO THE DP AND SEGMENTS G THROUGH A, RESPECTIVELY.
76			; IT IS POSSIBLE TO ACCOMMODATE
77			; DRIVERS WHICH ARE EITHER LOGICALLY INVERTING OR NON-INVERTING BY
78			; SETTING VARIABLE 'SEGPOL' (SEGMENT POLARITY).
79			; NOTE THAT BY HAVING ARBITRARY CONTROL OVER EACH SEGMENT, NON-NUMERIC
80			; CHARACTERS CAN BE REPRESENTED ON A SEVEN SEGMENT DISPLAY.
81			; AS SHOWN IN EXAMPLE SUBROUTINE 'TEST2'.
82			;
83			\$EJECT

```

84 ;PINPUT-FOUR HIGH-ORDER BITS USED AS INPUTS FROM THE KEYBOARD RETURN LINES
85 ; ASSUMES THAT A KEY DOWN IN THE CURRENTLY ENABLED ROW WOULD RETURN
86 ; A LOW LEVEL.
87 ; IN THIS CASE, BIT7 RETURNS THE LEFTMOST COLUMN, BIT4 THE RIGHTMOST.
88 ; THE HIGH-ORDER BITS ARE USED SO THAT IF AN OFF-CHIP DECODER IS USED
89 ; TO ENABLE UP TO 16 CHARACTERS, FOR EXAMPLE, IT COULD BE DRIVEN BY
90 ; THE LOW ORDER BITS OF THE SAME PORT.
91 ; NOTE ALSO THAT IF A SIXTEEN KEY MATRIX WERE ELECTRICALLY ORGANIZED
92 ; IN A 2X8 ARRAY, ONLY TWO RETURN LINES WOULD BE NEEDED.
93 ; (IN THIS CASE, PERHAPS T0 AND T1 COULD BE USED FOR INPUT BITS.)
94 ;
95 ;PULL-UP RESISTORS ON THE RETURN LINES MIGHT BE IN ORDER IF THERE IS ANY
96 ; POSSIBILITY OF A HIGH-IMPEDENCE CONDUCTIVE PATH THROUGH THE SWITCH WHEN
97 ; IT IS SUPPOSED TO BE 'OPEN'.
98 ; (THIS PHENOMENON HAS ACTUALLY BEEN OBSERVED.)
99 ;
100 ;THE DRIVERS USED IN THE PROTOTYPE WERE ALL NON-INVERTING IN THAT
101 ; A HIGH LEVEL ON AN OUTPUT LINE IS USED TO TURN A CHARACTER OR SEGMENT ON.
102 ; THERE ARE A TOTAL OF SEVEN I/O LINES LEFT OVER.
103 ;
104 ;THE ALGORITHM FOR DRIVING THE DISPLAY USES A BLOCK OF INTERNAL RAM
105 ; AS DISPLAY REGISTERS, WITH ONE BYTE CORRESPONDING TO EACH CHARACTER OF THE
106 ; DISPLAY. THE EIGHT BITS OF EACH BYTE CORRESPOND TO THE SEVEN SEGMENTS & DP
107 ; OF EACH CHARACTER. IF AN EXTERNAL ENCODER IS USED (SUCH AS A FOUR-BIT TO
108 ; SEVEN-SEGMENT ENCODER OR A ROM FOR TRANSLATING ASCII TO
109 ; SIXTEEN-SEGMENT "STARBURST" DISPLAY PATTERNS), THE TABLE ENTRIES WOULD HOLD
110 ; THE CHARACTER CODES. (IN THE FORMER CASE, AN UNUSED BIT COULD BE USED TO
111 ; ENABLE THE D.P.)
112 ;THUS, WRITING CHARACTERS TO THE DISPLAY FROM THE BACKGROUND PROGRAM
113 ; REALLY ENTAILS WRITING THE APPROPRIATE SEGMENT
114 ; PATTERNS TO A DISPLAY REGISTER- THE ACTUAL OUTPUTTING IS AUTOMATIC.
115 ; THE LEFTMOST CHARACTER CORRESPONDS TO THE LAST BYTE OF THE DISPLAY
116 ; REGISTERS, AND IS ACCESSED BY NEXTPL=8 (SEE SOURCE); THE RIGHTMOST
117 ; CHARACTER IS THE FIRST DISPLAY BYTE, WHEN NEXTPL=1.
118 ;UTILITY SUBROUTINES ARE INCLUDED HERE TO TRANSLATE FOUR BIT NUMBERS TO HEX
119 ; DIGIT PATTERNS, AND WRITE THEM INTO THE DISPLAY REGISTERS SEQUENTIALLY
120 ; (EITHER FILLING FROM THE LEFT- H.P. CALCULATOR STYLE OR FROM THE
121 ; RIGHT- T.I. STYLE; SUBROUTINES WDISP AND RENTRY, RESPECTIVELY).
122 ;
123 ;THE KEYBOARD SCANNING ALGORITHM SHOWN HERE REQUIRES A KEY BE DOWN FOR
124 ; SOME NUMBER OF COMPLETE DISPLAY SCANS TO BE ACKNOWLEDGED. SINCE IT IS
125 ; INTENDED FOR ONE-FINGER OPERATION, TWO-KEY ROLLOVER/N-KEY LOCKOUT HAS
126 ; BEEN IMPLEMENTED. HOWEVER, MODIFICATIONS WOULD BE POSSIBLE TO ALLOW, FOR
127 ; EXAMPLE, ONE KEY IN THE MATRIX TO BE USED AS A SHIFT KEY OR CONTROL KEY
128 ; TO BE HELD DOWN WHILE ANOTHER KEY IN THE MATRIX IS PRESSED. (SEE NOTE WITHIN
129 ; THE BODY OF THE LISTING.)
130 ;
131 $EJECT

```

```

LOC  OBJ      SEQ      SOURCE STATEMENT      ADDRESS
132 ; (BE AWARE THAT NO MORE THAN TWO KEYS CAN EVER BE DOWN UNLESS DIODES
133 ; ARE PLACED IN SERIES WITH ALL OF THE SWITCHES- CERTAINLY NOT THE CASE FOR EL
134 ; CHEAPO KEYBOARDS- BECAUSE SOME COMBINATIONS OF THREE KEYS DOWN WILL RESULT
135 ; IN A 'PHANTOM' FOURTH KEY BEING PERCEIVED.
136 ; THE PHANTOM KEY WOULD BE THE FOURTH 'CORNER' WHEN THREE KEYS FORMING
137 ; A RECTANGULAR PATTERN (IN THE X-Y KEY MATRIX) ARE DOWN.
138 ; IF DIODES ARE PLACED IN THE SCANNING ARRAY, CONSIDERATIONS MUST BE MADE
139 ; ABOUT HOW THE DIODE VOLTAGE DROP WILL AFFECT INPUT LOGIC LEVELS.
140 ;
141 ; WHEN A DEBOUNCED KEY IS DETECTED, THE NUMBER OF ITS POSITION IN THE KEY
142 ; MATRIX (LEFT-TO-RIGHT, BOTTOM-TO-TOP, STARTING FROM 00) IS PLACED INTO
143 ; RAM LOCATION 'KBDBUF'. AN INPUT SUBROUTINE THEN NEED ONLY READ THIS LOCATION
144 ; REPEATEDLY TO DETERMINE WHEN A KEY HAS BEEN PRESSED. WHEN A KEY IS DETECTED,
145 ; A SPECIAL CODE BYTE SHOULD BE WRITTEN BACK TO INTO 'KBDBUF' TO PREVENT
146 ; REPEATED DETECTIONS OF THE SAME KEY.
147 ; THE ROUTINE 'KBDIN' DEMONSTRATES A TYPICAL INPUT PROTOCOL, ALONG WITH A METHOD
148 ; FOR TRANSLATING A KEY POSITION TO ITS ASSOCIATED SIGNIFICANCE BY ACCESSING
149 ; TABLE 'LEGND$' IN ROM.
150 ;
151 $EJECT

```


LOC	OBJ	SEQ	SOURCE STATEMENT	OBJECT STATEMENT
0000		152	*****	
		153	ARE PLACED IN SERIES WITH ALL OF THE SWITCHES - CERTAIN THE CASE FOR B	
		154	INITIAL EQUATES TO DEFINE SYSTEM CONFIGURATION	
		155	;	
		156	*****	
		157	IN THE X-Y MATRIX ARE DOWN	
0010		158	PDIGIT EQU 00H ; USED TO ENABLE CHARACTERS AND STROBE ROWS OF KEYBOARD	
0008		159	PSGMINT EQU 00H ; USED TO TURN ON SEGMENTS OF CURRENTLY ENABLED DIGIT	
0009		160	PINPUT EQU P2 ; PORT USED TO SCAN FOR KEY CLOSURES	
		161	NOTE THAT THIS PORT ALLOCATION USES THE HIGHER	
		162	CURRENT SOURCING ABILITY OF THE BUS TO SWITCH ON THE	
		163	DIGIT DRIVERS AND LEAVES P23-P20 FREE FOR USING	
		164	AN 8243 PORT EXPANDER IN THE SYSTEM	
		165	A SPECIAL CODE BYTE SHOULD BE WRITTEN BACK TO INTO KEYBOARD TO PREVENT	
0000		166	POSLOG EQU 00H ; REPEATED DETECTION OF THE SAME KEY	
		167	NEGLOG EQU 0FFH ; THE ROUTINE 'KEYDOWN' INITIATES A KEY POSITION TO ITS ASSOCIATED	
0000		169	CHARPOL EQU POSLOG ; DEFINES WHETHER OUTPUT LINES ARE ACTIVE HI OR LOW	
0000		170	SEGPOL EQU POSLOG ; FOR DRIVING CHARACTERS AND SEGMENT PATTERNS	
00F0		171	INPMASK EQU 0F0H ; DEFINES BITS USED AS INPUT	
		172	;	
0008		173	CHARNO EQU 8 ; NUMBER OF DIGITS IN DISPLAY	
0004		174	NROWS EQU 4 ; ROWS OF KEYS (LESS THAN OR EQUAL TO CHARNO)	
0004		175	NCOLS EQU 4 ; LESSER DIMENSION OF KEYBOARD MATRIX	
		176	;	
FFF0		177	TICK EQU -10H ; DETERMINES INTERRUPT INTERVAL	
0004		178	DEENCE EQU 4 ; NUMBER OF SUCCESSIVE SCANS BEFORE KEY CLOSURE ACCEPTED	
0000		179	BLANK EQU 00H ; CODE TO BLANK DISPLAY CHARACTERS	
		180	;	
		181	;	
		182	;	
000F		183	ENCMASK EQU 0FH ; SELECTS WHICH BITS ARE RELEVANT TO ENCCACC SUBROUTINE	
		184	;	
		185	EJECT	

LOC	OBJ	SEQ	SOURCE STATEMENT
		186	*****
		187	*****
		188	BANK 0 REGISTERS USED
		189	*****
		190	POINTERS USED FOR INDIRECT RAM ACCESSING:
0000		191	PNTR0 EQU R0
0001		192	PNTR1 EQU R1
0007		193	NEXTPL EQU R7 ; USED TO KEEP TRACK OF CHARACTER POSITION BEING
		194	; WRITTEN INTO
		195	*****
		196	*****
		197	*****
		198	BANK 1 REGISTER ALLOCATION
		199	*****
		200	PNTR0 EQU R0 (ALREADY DEFINED)
		201	PNTR1 EQU R1
0002		202	ASAVE EQU R2 ; HOLDS ACCUMULATOR VALUE DURING SERVICE ROUTINE
0004		203	ROTPAT EQU R4 ; USED TO HOLD INPUT PATTERN BEING ROTATED THROUGH CY
0005		204	ROTCNT EQU R5 ; COUNTS NUMBER OF BITS ROTATED THROUGH CY
0006		205	LASTKY EQU R6 ; HOLDS KEY POSITION OF LAST KEY DEPRESSION DETECTED
0007		206	CURDIG EQU R7 ; HOLDS POSITION OF NEXT CHARACTER TO BE DISPLAYED
		207	*****
		208	*****
		209	*****
		210	DATA RAM ALLOCATION
		211	*****
0020		212	NREPTS EQU 32 ; KEEPS TRACK OF SUCCESSIVE READS OF SAME KEYSTROKE
0021		213	KEYLOC EQU 33 ; INCREMENTED AS SUCCESSIVE KEY LOCATIONS SCANNED
0022		214	KDBBUF EQU 34 ; CARRIES POSITION OF DEBOUNCED KEY FROM REFRSH ROUTINE
		215	; \ BACK TO BACKGROUND PROGRAM
0023		216	RDELAY EQU 35 ; NON-ZERO WHEN DISPLAY IN PROGRESS
		217	*****
		218	THE LAST <CHARNO> REGISTERS HOLD THE DISPLAY SEGMENT PATTERNS
		219	*****
0037		220	SEGMAP EQU (63-CHARNO) ; BASE OF REGISTER ARRAY FOR DISPLAY PATTERNS
		221	; \ (COULD BE ANYWHERE IN INTERNAL RAM)
		222	*****
		223	*****
		224	*****
		225	NOTE THAT LASTKY, CURDIG, AND F1 RETAIN STATUS INFORMATION FROM
		226	ONE INTERRUPT TO THE NEXT. ALL OTHER REGISTERS MAY BE USED IN
		227	THE USER'S OWN INTERRUPT SERVICING ROUTINE
		228	*****
		229	*****
		230	*****
		231	\$EJECT

```

0000 233 ; *****
0000 0460 234 ;
235 ORG 000H
236 JMP INIT
237 ;
238 ;
239 ; *****
0007 240 ;
241 ORG 007H
242 ;
243 ; TIINT TIMER INTERRUPT SUBROUTINE
244 ; CALL MADE TO LOC. 007H WHEN TIMER TIMES OUT.
245 ; TIMER CAN BE RE-INITIALIZED AT THIS POINT IF DESIRED.
246 ; USED HERE TO CAUSE THE DISPLAY REFRESH AND KEY SCAN ROUTINES TO
247 ; BE CALLED PERIODICALLY.
0007 D5 248 TIINT: SEL RB1
0008 AA 249 MOV ASAVE, A
0009 23F0 250 MOV A, #TICK
000B 62 251 MOV T, A ; RELOAD TIMER INTERVAL
252 ;
253 ; *****
254 ;
255 ; THE USER'S OWN TIMER INTERRUPT ROUTINE (IF IT EXISTS) COULD
256 ; BE PLACED AT THIS POINT.
257 ;
258 ; *****
000C 1410 259 ;
260 CALL REFRESH ; CAUSE DISPLAY TO BE UPDATED
261 ;
262 ; THE COMPLETE INTERRUPT ROUTINE SHOULD BE COPIED HERE
263 ; TO SAVE A FULL LEVEL OF SUBROUTINE NESTING.
264 ; IT WAS WRITTEN AS A SUBROUTINE HERE FOR THE SAKE OF CLARITY.
265 ;
266 ; *****
267 ;
268 ; TIRET TIMER INTERRUPT RETURN CODE- RESTORES ACC VALUE
000E FA 269 TIRET: MOV A, ASAVE
000F 93 270 RETR
271 ;
272 $EJECT

```

LOC	OBJ	SEQ	SOURCE STATEMENT
		273	*****
		274	REFRSH SUBROUTINE TO MULTIPLEX SEVEN-SEGMENT DISPLAYS.
		275	EACH CALL CAUSES THE NEXT CHARACTER TO BE DISPLAYED,
		276	ACCORDING TO THE CONTENTS OF THE SEGMAP REGISTER ARRAY.
		277	REFRSH SHOULD BE CALLED AT LEAST EVERY MSEC OR SO.
		278	*****
		279	
0010	2300	280	REFRSH: MOV A, #BLANK XOR SEGPOL
0012	39	281	OUTL PSGMNT, A ;WRITE BLANK PATTERN TO SEG DRIVERS
0013	2357	282	REFR1: MOV A, #CHRSTB ;LOOK UP DIGIT-ENABLE PATTERN
0015	6F	283	ADD A, CURDIG ;ADD CURDIG DISPLACEMENT
0016	A3	284	MOVP A, @A ;ENABLE ONE BIT OF ACCUMULATOR
0017	02	285	OUTL PDIGIT, A ;ENERGIZE CHARACTER
		286	
		287	
0018	2337	288	MOV A, #SEGMAP ;LOAD BASE OF REGISTER ARRAY
001A	6F	289	ADD A, CURDIG ;ADD CURDIG DISPLACEMENT
001B	A9	290	MOV PNTR1, A
001C	F1	291	MOV A, @PNTR1 ;LOAD ACC W/ NEXT SEGMENT PATTERN
001D	39	292	OUTL PSGMNT, A ;ENABLE APPROPRIATE SEGMENTS
		293	
		294	*****
		295	THE NEXT CHARACTER IS NOW BEING DISPLAYED.
		296	THE KEYBOARD SCAN ROUTINE IS INTEGRATED INTO THE DISPLAY SCAN.
		297	WITH THE CURRENT ROW ENERGIZED, CHECK IF THERE ARE ANY INPUTS.
		298	*****
		299	
001E	B821	300	SCAN: MOV PNTR0, #KEYLOC ;SET POINTER FOR SEVERAL KEYLOC REFERENCES
0020	0A	301	IN A, PINPUT ;LOAD ANY SWITCH CLOSURES
		302	
		303	*****
		304	## THIS BLOCK OF CODE IS NOT NEEDED BY THE KEYBOARD SCAN LOGIC. ##
		305	## HOWEVER, ITS INCLUSION WOULD SPEED THINGS UP A BIT BY ##
		306	## SKIPPING OVER ROWS IN WHICH NO KEYS ARE DOWN. ##
		307	## IT WAS OMITTED HERE TO CONSERVE ROM SPACE, BUT MIGHT BE ##
		308	## RESTORED IF VERY LARGE KEYBOARDS (ESPECIALLY THOSE WITH EIGHT ##
		309	## KEYS PER ROW) ARE TO BE USED WITH THIS ALGORITHM. ##
		310	*****
		311	## CPL A ;ANY CLOSURES DETECTED ARE NOW ONE BITS ##
		312	## ANL A, #INPMASK ##
		313	## JNZ SCAN1 ;-IF A KEY IN THE CURRENTLY ENABLED ROW IS DOWN ##
		314	## NO KEY IS NOW DOWN SO THE KEYLOC COUNT MAY BE UPDATED DIRECTLY ##
		315	## MOV A, @PNTR0 ##
		316	## ADD A, #NCOLS ##
		317	## MOV @PNTR0, A ##
		318	## JMP SCAN6 ##
		319	*****
		320	## IF THIS CODE IS USED, SUBSTITUTE THE 'JC SCAN5' FOUR LINES ##
		321	## HENCE WITH 'JNC SCAN5' TO ACCOMODATE THE INVERTED POLARITY ##
		322	*****
		323	\$EJECT

```

LOC OBJ      SEQ      SOURCE STATEMENT
-----
324 ; *****
325 ROTATE BITS THROUGH THE CY WHILE INCREMENTING KEYLOC.
326 ; *****
327 ; *****
0021 BD04     328 SCAN1: MOV     ROTCNT, #NCOLS ;SET UP FOR <NCOLS> LOOPS THROUGH 'NXTLOC'
0023 F7       329 NXTLOC: RLC     A
0024 AC       330 MOV     ROTPAT, A ;SAVE SHIFTED BIT PATTERN
0025 F63F     331 JC      SCANS ;ONE BIT IN CY INDICATES KEY NOT DOWN
332 ; *****
333 ; *****
334 ; *****
335 ; AT THIS POINT IT HAS JUST BEEN DETERMINED THAT THE VALUE
336 ; OF KEYLOC IS THE POSITION OF A KEY WHICH IS NOW DOWN.
337 ; THE FOLLOWING CODE DEBOUNCES THE KEY, ETC.
338 ; IF MODIFICATIONS TO THE KEYBOARD LOGIC, I.E. THE INCLUSION
339 ; OF A SHIFT, CONTROL, OR MODE KEY IN THE KEY MATRIX ITSELF)
340 ; ARE DESIRED, THEY SHOULD BE MADE AT THIS POINT, BEFORE
341 ; THE DEBOUNCE LOGIC BEGINS. FOR EXAMPLE, AT THIS POINT
342 ; KEYLOC COULD BE COMPARED AGAINST THE POSITION OF THE MODE
343 ; KEY, AND IF THEY MATCH SET SOME FLAG BIT AND JUMP TO
344 ; LABEL 'SCANS'. OR, BY COMPARING KEYLOC AGAINST THE LAST
345 ; KEY DEBOUNCED, IMMEDIATE TWO-KEY ROLLOVER COULD BE
346 ; IMPLEMENTED.
347 ; *****
348 ; *****
349 ; *****
0027 A5       350 CLR     F1 ;MARK THAT AT LEAST ONE KEY WAS DETECTED
0028 B53F     351 CPL     F1 ;\ IN THE CURRENT SCAN
352 ; *****
353 ; *****
354 ; A KEYSTROKE WAS DETECTED FOR THE CURRENT COLUMN. ITS
355 ; POSITION IS IN REGISTER KEYLOC. SEE IF SAME KEY SENSED LAST CYCLE.
356 ; *****
357 ; *****
0029 F0       358 MOV     A, @PNTR0 ;PNTR0 STILL HOLDS #KEYLOC
002A 2E3F     359 XCH     A, LASTKY
002B DE       360 XRL     A, LASTKY
002C B820     361 MOV     PNTR0, #NREPTS ;PREPARE TO CHECK AND/OR MODIFY REPEAT COUNT
002E C634     362 JZ      SCANS
363 ;
364 $EJECT

```


LOC	OBJ	SEQ	SOURCE STATEMENT	THIRDTAT2	SCANS2	032	180	COL
		365	*****					
		366	***** A DIFFERENT KEY WAS READ ON THIS CYCLE THAN ON THE PREVIOUS CYCLE.					
		367	***** SET NREPTS TO THE DEBOUNCE PARAMETER FOR A NEW COUNTDOWN.					
		368	*****					
		369	*****					
0030	B004	370	MOV @PNTR0, #DEBNCE					
0032	043F	371	JMP SCAN5					
		372	*****					
		373	*****					
		374	***** SAME KEY WAS DETECTED AS ON PREVIOUS CYCLE					
		375	***** LOOK AT NREPTS: IF ALREADY ZERO, DO NOTHING.					
		376	***** ELSE DECREMENT NREPTS.					
		377	***** IF THIS RESULTS IN ZERO, MOVE LASTKY INTO KDBUF.					
		378	*****					
		379	*****					
0034	F0	380	SCAN3: MOV A, @PNTR0					
0035	C63F	381	JZ SCAN5 ; IF ALREADY ZERO					
0037	07	382	DEC A ; INDICATE ONE MORE SUCCESSIVE KEY DETECTION					
0038	A0	383	MOV @PNTR0, A					
0039	963F	384	JNZ SCAN5 ; IF DECREMENT DOES NOT RESULT IN ZERO					
003B	FE	385	MOV A, LASTKY					
003C	B822	386	MOV PNTR0, #KDBUF					
003E	A0	387	MOV @PNTR0, A ; TO MARK NEW KEY CLOSURE					
		388	*****					
003F	B821	389	SCAN5: MOV PNTR0, #KEYLOC					
0041	10	390	INC @PNTR0					
0042	FC	391	MOV A, ROTPAT					
0043	ED23	392	DJNZ ROTCNT, NXTLOC					
		393	*****					
		394	*****					
0045	EF57	395	SCAN6: DJNZ CURDIG, SCAN9					
		396	*****					
		397	#EJECT					

```

398 ; *****
399 ; *****
400 ; ***** THE FOLLOWING CODE SEGMENT IS USED BY THE KEYBOARD SCANNING ROUTINE.
401 ; ***** IT IS EXECUTED ONLY AFTER A REFRESH SEQUENCE OF ALL
402 ; THE CHARACTERS IN THE DISPLAY IS COMPLETED.
403 ; *****
404 ;
405 MOV CURDIG, #CHARNO
406 MOV @PNTR0, #0 ; PNTR0 STILL CONTAINS #KEYLOC
407 JF1V39A SCAN8 ; JUMP IF ANY KEYS WERE DETECTED
408 MOV LASTKY, #0FFH ; CHANGE <LASTKY> WHEN NO KEYS ARE DOWN
409 SCAN8: CLR F1
410 ; *****
411 ; *****
412 ; THE NEXT CODE SEGMENT IS THE INTERRUPT-DRIVEN PORTION OF THE 'DELAY'
413 ; UTILITY. IT DECREMENTS RAM LOCATION 'RDELAY' ONCE PER DISPLAY SCAN
414 ; IF 'RDELAY' IS NOT ALREADY ZERO.
415 ; *****
416 ;
417 MOV PNTR1, #RDELAY
418 MOV A, @PNTR1
419 JZ SCAN9
420 DEC A
421 MOV @PNTR1, A
422 ;
423 SCAN9: RET
424 ;
425 ; *****
426 ;
427 ; CHRSTB IS THE BASE FOR THE PATTERNS TO ENABLE ONE-OF-CHARNO CHARACTERS.
428 CHRSTB EQU ($-1) AND 0FFH
429 DB (00000010 XOR CHRSTB)
430 DB (00000010 XOR CHRSTB)
431 DB (00000100 XOR CHRSTB)
432 DB (00001000 XOR CHRSTB)
433 DB (00010000 XOR CHRSTB)
434 DB (00100000 XOR CHRSTB)
435 DB (01000000 XOR CHRSTB)
436 DB (10000000 XOR CHRSTB)
437 ;
438 $EJECT

```



```

LOC  OBJ      SEQ      SOURCE STATEMENT      TARGET STATEMENT      PC      SP      BP
-----
476 ; *****
477 ;
478 ; THE FOLLOWING SUBROUTINES IMPLEMENT THE UTILITIES COMMONLY USED FOR
479 ; MOST KEYBOARD/DISPLAY APPLICATIONS.
480 ; THEY COULD BE USED EXACTLY AS SHOWN HERE OR ADAPTED FOR SPECIAL CASES.
481 ;
482 ; *****
483 ;
234 484 ; KBDIN: KEYBOARD INPUT SUBROUTINE.
485 ; COULD BE USED TO INTERFACE THE USER'S BACKGROUND PROGRAM WITH
293 486 ; THE INTERRUPT DRIVEN KEYBOARD SCANNER.
487 ; RETURNS ONLY AFTER A NEW KEYSTROKE HAS BEEN DETECTED AND DEBOUNCED.
488 ; ENCODED VALUE OF KEY (RATHER THAN ITS POSITION IN SWITCH MATRIX) IS
489 ; RETURNED IN THE ACCUMULATOR.
0083 490 KBDIN: MOV     PNTR1, #KBDGUF
0085 491 MOV     A, #00H ; KBDGUF WILL BE MARKED AS CLEAR
0087 492 XCH     A, @PNTR1 ; LOAD BUFFER VALUE
0088 493 JB7     KBDIN
008A 494 ADD     A, #LEGND5 ; ADD BASE OF KEY ENCODING TABLE
008C 495 MOVP    A, @A ; OBTAIN BYTE REPRESENTING KEY SIGNIFICANCE
008D 496 RET
497 ;
498 ;
499 ; LEGND5 IS THE BASE FOR TABLE SHOWING KEY MATRIX SIGNIFICANCE
500 ; FOR THE KEYBOARD USED IN THE PROTOTYPE.
501 ; KEY LAYOUT IS AS SHOWN TO THE RIGHT.
502 ;
503 ; NOTE THAT BIT6-BIT4 MAY BE USED TO ENCODE KEY TYPE. IN THIS CASE:
504 ; BIT4 INDICATES REGULAR DECIMAL DIGITS.
505 ; BIT5 INDICATES RIGHT-COLUMN FUNCTION KEYS.
506 ; BIT6 INDICATES PUNCTUATION MARKS ( * AND # ).
008E 507 EQU     EQU, 00H ($ AND 0FFH) ; USE LOW ORDER BITS AS TABLE INDEX
008E 4F 508 DB     4FH
008F 10 509 DB     00H
0090 4E 510 DB     00H
0091 28 511 DB     28H ; PDIGIT4==> 1 2 3 4 <1>
0092 17 512 DB     17H
0093 18 513 DB     18H ; PDIGIT5==> 4 5 6 <2>
0094 19 514 DB     19H
0095 24 515 DB     24H ; PDIGIT6==> 7 8 9 <3>
0096 14 516 DB     14H
0097 15 517 DB     15H ; PDIGIT7==> * 0 # <4>
0098 16 518 DB     16H
0099 22 519 DB     22H ;
009A 11 520 DB     11H ;
009B 12 521 DB     12H ;
009C 13 522 DB     13H ;
009D 21 523 DB     21H ;
524 DB     21H
525 $EJECT

```

LOC	OBJ	SEQ	SOURCE STATEMENT	THIRGSTATE	THIRGSTATE	LOC	OBJ
		526	*****				
		527					
		528	CLEAR: WRITES 'BLANK' CHARACTERS INTO ALL DISPLAY REGISTERS.				
		529					
		530	FILL: RETURNS WITH NEXTPL SET TO LEFTMOST CHARACTER POSITION				
009E	2300	531	CLEAR: MOV A, #BLANK XOR SEGPOL				
00A0	B938	532	FILL: MOV PNTR1, #SEGMAP+1				
00A2	BF08	533	MOV NEXTPL, #CHARNO				
00A4	A1	534	CLR1: MOV @PNTR1, A				
00A5	19A8	535	INC PNTR1: POINT TO NEXT CHARACTER TO THE LEFT				
00A6	EFA4	536	DJNZ NEXTPL, CLR1				
00A8	BF08	537	MOV NEXTPL, #CHARNO				
00AA	83	538	RET				
		539					
		540	*****				
		541					
		542	PRINT: SUBROUTINE TO COPY A STRING OF BIT PATTERNS FROM ROM TO THE				
		543	DISPLAY REGISTERS. STRING STARTS AT LOCATION POINTED TO BY PNTR0				
		544	CONTINUES UNTIL AN ESCAPE CODE (0FFH) IS REACHED.				
		545	NOTE THAT THE CHARACTER STRING PUT OUT MUST BE LOCATED ON THE SAME				
		546	PAGE AS THIS SUBROUTINE, SINCE SAME-PAGE MOVES ARE USED.				
		547	PRINT IN TURN CALLS EITHER SUBROUTINE 'WDISP' OR 'RENTY'				
		548	TO ACTUALLY EFFECT WRITING INTO THE DISPLAY REGISTERS.				
00AB	F8	549	PRINT: MOV A, PNTR0				
00AC	A3	550	MOVP A, @A				
00AD	C6B4	551	JZ PRNT1: ESCAPE PATTERN				
00AF	14D0	552	CALL WDISP: OUTPUT TO NEXT CHARACTER POSITION				
		553	##: CALL RENTRY: INSTEAD IF MESSAGE IS TO BE RIGHT JUSTIFIED)				
00B1	18	554	INC PNTR0: INDEX POINTER				
00B2	04AB	555	JMP PRINT				
00B4	83	556	PRNT1: RET				
		557					
		558	*****				
		559					
		560	JOHN: ARRAY HOLDS THE BIT PATTERNS FOR THE LETTERS 'JOHN' (SEE 'TEST2')				
		561	(NOTE THAT 'JOHN' IS WRITTEN IN LOWER CASE LETTERS)				
00B5		562	JOHN: EQU \$ AND 0FFH				
00B5	1E	563	DB 00011110B XOR SEGPOL				
00B6	5C	564	DB 01011100B XOR SEGPOL				
00B7	74	565	DB 01101000B XOR SEGPOL				
00B8	54	566	DB 01010100B XOR SEGPOL				
00B9	00	567	DB 00				
		568					
		569	EJECT				

LOC	OBJ	SEQ	SOURCE STATEMENT
		570	*****
		571	;
		572	ENCACC ENCODES LSNIBBLE OF ACC INTO HEX; BIT PATTERN INTO ACC
00BA 530F		573	ENCACC: ANL A, #ENCMASK
00BC 03C0		574	ADD A, #DGPATS
00BE A3		575	MOV A, @A
00BF 83		576	RET
		577	DGPATS IS THE BASE FOR THE TABLE OF SEGMENT PATTERNS FOR THE BASIC
		578	DIGITS. HERE THE FULL HEX SET (0-F) IS INCLUDED.
		579	FOR MANY USER APPLICATIONS, THE CHARACTER SET MAY BE AMENDED OR AUGMENTED
		580	TO INCLUDE ADDITIONAL SPECIAL PURPOSE PATTERNS.
		581	FORMAT IS PGFEDCBA IN STANDARD SEVEN-SEGMENT ENCODING CONVENTION
		582	WHERE P REPRESENTS THE DECIMAL POINT
00C0		583	DGPATS EQU \$ AND 0FFH
00C0 3F		584	DB 00111111B XOR SEGPOL
00C1 06		585	DB 00000110B XOR SEGPOL
00C2 5B		586	DB 01011011B XOR SEGPOL
00C3 4F		587	DB 01001111B XOR SEGPOL
00C4 66		588	DB 01100110B XOR SEGPOL
00C5 6D		589	DB 01101101B XOR SEGPOL
00C6 7D		590	DB 01111101B XOR SEGPOL
00C7 07		591	DB 00000111B XOR SEGPOL
00C8 7F		592	DB 01111111B XOR SEGPOL
00C9 67		593	DB 01100111B XOR SEGPOL
00CA 77		594	DB 01110111B XOR SEGPOL
00CB 7C		595	DB 01111100B XOR SEGPOL
00CC 39		596	DB 00111001B XOR SEGPOL
00CD 5E		597	DB 01011100B XOR SEGPOL
00CE 79		598	DB 01111001B XOR SEGPOL
00CF 71		599	DB 01110001B XOR SEGPOL
		600	;
		601	*****
		602	;
		603	WDISP WRITES BIT PATTERN NOW IN ACC INTO NEXT CHARACTER POSITION
		604	OF THE DISPLAY (NEXTPL). ADJUSTS NEXTPL POINTER VALUE.
		605	RESULTS IN DISPLAY BEING FILLED LEFT TO RIGHT, THEN RESTARTING
00D0 A9		606	WDISP: MOV PNTR1, A
00D1 FF		607	MOV A, NEXTPL
00D2 0337		608	ADD A, #SEGMAP
00D4 29		609	XCH A, PNTR1
00D5 A1		610	MOV @PNTR1, A
00D6 EFDA		611	DJNZ NEXTPL, WDISP1
00D8 BF08		612	MOV NEXTPL, #CHARNO
00DA 83		613	WDISP1: RET
		614	;
		615	\$EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT
		616	;*****
		617	;
		618	;RENTY SUBROUTINE TO ENTER ACC CONTENTS INTO THE RIGHTMOST DIGIT
		619	; AND SHIFT EVERYTHING ELSE ONE PLACE TO THE LEFT
00DB	B938	620	RENTY: MOV PNTR1, #SEGMAP+1
00DD	BF08	621	MOV NEXTPL, #CHARNO
00DF	21	622	RENTY: XCH A, @PNTR1
00E0	19	623	INC PNTR1
00E1	EFDF	624	DJNZ NEXTPL, RENTY
00E3	BF08	625	MOV NEXTPL, #CHARNO ;POINT TO LEFTMOST CHARACTER
00E5	83	626	RET
		627	;
		628	;*****
		629	;
		630	;RDPADD TOGGLE DECIMAL POINT IN LAST CHARACTER DISPLAY CHARACTER
		631	;DPADD TOGGLES DECIMAL POINT IN THE CHARACTER POINTED TO BY THE ACC
		632	;
00E6	2301	633	RDPADD: MOV A, #01H ;SET INDEX TO RIGHTMOST POSITION
00E8	0337	634	DPADD: ADD A, #SEGMAP ;ACCESS DISPLAY REGISTER FOR DESIRED PLACE
00EA	A9	635	MOV PNTR1, A
00EB	F1	636	MOV A, @PNTR1
00EC	D300	637	XRL A, #00H
00EE	A1	638	MOV @PNTR1, A
00EF	83	639	RET
		640	;
		641	;*****
		642	;
		643	;HOLD SUBROUTINE CALLED WHEN KEY IS KNOWN TO BE DOWN.
		644	; WILL NOT RETURN UNTIL KEY IS RELEASED.
00F0	D5	645	HOLD: SEL RB1
00F1	FE	646	MOV A, LASTKY ;(LASTKY)=0FFH IFF NO KEYS DOWN
00F2	C5	647	SEL RB0
00F3	37	648	CPL A
00F4	96F0	649	JNZ HOLD
00F6	83	650	RET
		651	;
		652	;*****
		653	;
		654	;DELAY SUBROUTINE HANGS UP FOR THE NUMBER OF COMPLETE DISPLAY SCANS, EQUAL
		655	; TO THE CONTENTS OF THE ACCUMULATOR WHEN CALLED.
00F7	B923	656	DELAY: MOV PNTR1, #RDELAY
00F9	A1	657	MOV @PNTR1, A
00FA	F1	658	DELAY1: MOV A, @PNTR1
00FB	96FA	659	JNZ DELAY1
00FD	83	660	RET
		661	\$EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT
0100		662	ORG 100H
		663	;
		664	*****
		665	;
		666	THE CODE ON THIS PAGE IS FOR DEMONSTRATION PURPOSES ONLY-
		667	I TRULY DOUBT WHETHER ANY END USERS WOULD LIKE TO SEE A NAME
		668	POPPING UP ON THEIR CALCULATOR SCREENS
		669	HOWEVER, THE CODE SHOWN HERE DOES INDICATE HOW THE UTILITY SUBROUTINES
		670	INCLUDED HERE COULD BE ACCESSED.
		671	THE ROUTINES THEMSELVES ARE CALLED WHEN ONE OF THE FOUR BUTTONS
		672	ON THE RIGHT-HAND SIDE OF THE PROTOTYPE KEYBOARD IS PRESSED.
		673	;
		674	*****
		675	;
		676	FUNCTN ROUTINE TO IMPLEMENT ONE OF FOUR DEMO UTILITIES, ACCORDING
		677	TO WHICH OF THE FOUR FUNCTION KEYS WAS PRESSED
0100	1212	678	FUNCTN: JB0 FUNCT1
0102	320E	679	JB1 FUNCT2
0104	520A	680	JB2 FUNCT3
		681	;
0106	14E6	682	FUNCT4: CALL RDPADD
0108	0477	683	JMP ECHO
		684	;
010A	342E	685	FUNCT3: CALL TEST3
010C	0477	686	JMP ECHO
		687	;
010E	3424	688	FUNCT2: CALL TEST2
0110	0477	689	JMP ECHO
		690	;
0112	3416	691	FUNCT1: CALL TEST1
0114	0477	692	JMP ECHO
		693	;
		694	*****
		695	;
		696	TEST1 CODE SEGMENT TO FILL DISPLAY REGISTERS WITH DIGITS DOWN TO '1'
0116	BF08	697	TEST1: MOV NEXTPL, #CHARNO
0118	B808	698	MOV PNTR0, #CHARNO ;SET FOR EIGHT LOOP REPETITIONS
011A	FF	699	TST11: MOV A, NEXTPL
011B	148A	700	CALL ENCC
011D	14D0	701	CALL WDISP
011F	E81A	702	DJNZ PNTR0, TST11 ;COPY NEXT DIGIT INTO DISPLAY REGISTERS
0121	BF08	703	MOV NEXTPL, #CHARNO
0123	83	704	RET
		705	;
		706	\$EJECT

0000	0001	0002	0003
009E CLR1 00A4	CURDIG 0007		
0077 ENCACC 00BA	ENCM5K 000F		
0106 FUNCTN 0100	HOLD 00F0		
0021 LASTKY 0006	LEGND5 000E		
0023 PDIGIT 0010	PINPUT 0009		
0008 RDELAY 0023	RDPADD 00E6		
0004 SCAN 001E	SCAN1 0021		
0037 SEGPOL 0000	TEST1 0116		
011A WDISP 0000	WDISP1 0000		

All trademarks copyrighted © Intel Corporation 1976.

ROTCNT	204#	328	392															
ROTPAT	203#	330	391															
SCAN	300#																	
SCAN1	328#																	
SCAN3	362	380#																
SCAN5	331	371	381	384	389#													
SCAN6	395#																	
SCAN8	407	409#																
SCAN9	395	419	423#															
SEGMAP	220#	288	532	608	620	634												
SEGPOL	170#	280	531	563	564	565	566	584	585	586	587	588	589	590	591	592		
	593	594	595	596	597	598	599	722										
TEST1	691	697#																
TEST2	688	711#																
TEST3	685	722#																
TICK	177#	250	451															
TIINT	248#																	
TIRET	269#																	
TST11	699#	702																
WDISP	552	606#	701															
WDISP1	611	613#																

CROSS REFERENCE COMPLETE

8049 Microcomputer

however, the high speed of the 8049 makes the performance of the microcomputer control with no hardware requirements beyond two of the I/O pins already resident on the microcomputer.

There are many techniques for implementing serial I/O. The application note "Application Software Control: The Application Note" describes the techniques for the MCS-48 Family, describes several alternatives suitable for full duplex operation. Full duplex operation is more difficult, however, since it requires the receiver and transmit processes to operate concurrently. This difficulty is made more severe if it is necessary for some other process to also operate while serial communication is occurring. Scanning a keyboard and display, for example, is a common operation of single chip microcomputer based system which might have to occur concurrently with the serial receiver/transmitter process. The next section will describe an algorithm which implements full duplex serial communication to occur concurrently with other tasks. The design goal was to allow 2400 baud, full duplex, serial communication while utilizing no more than 50% of the available processing power of the high speed 8049 microcomputer.

The format used for most asynchronous communication is shown in Figure 1. It consists of eight data bits with a leading 'START' bit and one or more trailing 'STOP' bits. The START bit is used to establish synchronization between the receiver and transmitter. The STOP bit ensures that the receiver will be ready to synchronize itself when the next start bit occurs. Two stop bits are normally used for 110 baud communication and one stop bit for higher rates.

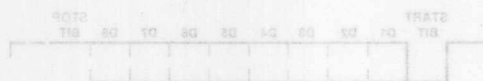


Figure 1

The algorithm used for reception of the serial data is shown in Figure 2. It uses the on board timer of the 8049 to establish a sampling period of four times the desired baud rate. For 2400 baud operation a crystal frequency of 8.218 MHz was chosen after the following calculation:

$$1 = 4800 / (2400 \times 4)$$

where 480 is the factor by which the crystal frequency is divided within the processor to get the basic interrupt rate
2400 is the desired baud rate
4 is the required number of samples per bit time
N is the value loaded into the MCS-48 timer when it overflows

INTRODUCTION	3-60
FULL DUPLEX SERIAL COMMUNICATIONS	3-60
MULTIPLY ALGORITHMS	3-69
DIVIDE ALGORITHMS	3-72
BINARY AND BCD CONVERSIONS	3-75
CONCLUSION	3-81

It is obvious that this increase in performance is going to result in far more ambitious programs being written for execution in a single chip microcomputer. This article will show how several program modules can be designed using the 8049. These modules were chosen to illustrate the capability of the 8049 in frequently encountered design situations. The modules included are: full duplex serial I/O, binary multiply and divide routines, binary to BCD conversions, and BCD to binary conversion. It should be noted that since the 8049 is totally software compatible with the 8748 and 8048 these routines will also be useful directly on these processors. In addition the algorithms for these programs are expressed in a program design language format which should allow them to be easily understood and extended to suit individual applications with minimal problems.

FULL DUPLEX SERIAL COMMUNICATIONS

Serial communications have always been an important facet in the application of microprocessors. Although this has been partially due to the necessity of connecting a terminal to the microprocessor based system for program generation and debug, the main impetus has been the simple fact that a large share of microprocessors find their way into end products (such as intelligent terminals) which themselves depend on serial communication. When it is necessary to add a serial link to a microprocessor such as the Intel® MCS-86 or 88 the solution is easy; the Intel® 8251A USART or 8232 SCLC chip can easily be added to provide the necessary protocol. When it is necessary to do the same thing to a single chip microcomputer, however, the situation becomes more difficult.

Some microcomputers, such as the Intel 8048 and 8049 have a complete bus interface built into them which allows the simple connection of a USART to the processor chip. Most other single chip microcomputers, although lacking such a bus, can be connected to a USART with various artificial hardware and software constructs. The difficulty with using these chips,

a single chip. The performance of the initial processors in the family (the 8748 and the 8048) has been shown to meet or exceed the requirements of most current applications of microcomputers. A new member of the family, however, has been recently introduced which promises to allow the use of the single chip microcomputer in many application areas which have previously required a multichip solution. The Intel® 8049 virtually doubles processing power available to the systems designer. Program storage has been increased from 1K bytes to 2K bytes, data storage has been increased from 64 bytes to 128 bytes, and processing speed has been increased by over 80%. (The 2.5 microsecond instruction cycle of the first members of the family has been reduced to 1.36 microseconds.)

It is obvious that this increase in performance is going to result in far more ambitious programs being written for execution in a single chip microcomputer. This article will show how several program modules can be designed using the 8049. These modules were chosen to illustrate the capability of the 8049 in frequently encountered design situations. The modules included are full duplex serial I/O, binary multiply and divide routines, binary to BCD conversions, and BCD to binary conversion. It should be noted that since the 8049 is totally software compatible with the 8748 and 8048 these routines will also be useful directly on these processors. In addition the algorithms for these programs are expressed in a program design language format which should allow them to be easily understood and extended to suit individual applications with minimal problems.

FULL DUPLEX SERIAL COMMUNICATIONS

Serial communications have always been an important facet in the application of microprocessors. Although this has been partially due to the necessity of connecting a terminal to the microprocessor based system for program generation and debug, the main impetus has been the simple fact that a large share of microprocessors find their way into end products (such as intelligent terminals) which themselves depend on serial communication. When it is necessary to add a serial link to a microprocessor such as the Intel® MCS-85 or 86 the solution is easy; the Intel® 8251A USART or 8273 SDLC chip can easily be added to provide the necessary protocol. When it is necessary to do the same thing to a single chip microcomputer, however, the situation becomes more difficult.

Some microcomputers, such as the Intel 8048 and 8049 have a complete bus interface built into them which allows the simple connection of a USART to the processor chip. Most other single chip microcomputers, although lacking such a bus, can be connected to a USART with various artificial hardware and software constructs. The difficulty with using these chips,

makes it feasible to implement a serial link under software control with no hardware requirements beyond two of the I/O pins already resident on the microcomputer.

There are many techniques for implementing serial I/O under software control. The application note "Application Techniques for the MCS-48 Family" describes several alternatives suitable for half duplex operation. Full duplex operation is more difficult, however, since it requires the receive and transmit processes to operate concurrently. This difficulty is made more severe if it is necessary for some other process to also operate while serial communication is occurring. Scanning a keyboard and display, for example, is a common operation of single chip microcomputer based system which might have to occur concurrently with the serial receive/transmit process. The next section will describe an algorithm which implements full duplex serial communication to occur concurrently with other tasks. The design goal was to allow 2400 baud, full duplex, serial communication while utilizing no more than 50% of the available processing power of the high speed 8049 microcomputer.

The format used for most asynchronous communication is shown in Figure 1. It consists of eight data bits with a leading 'START' bit and one or more trailing 'STOP' bits. The START bit is used to establish synchronization between the receiver and transmitter. The STOP bits ensure that the receiver will be ready to synchronize itself when the next start bit occurs. Two stop bits are normally used for 110 baud communication and one stop bit for higher rates.

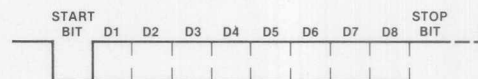


Figure 1.

The algorithm used for reception of the serial data is shown in Figure 2. It uses the on board timer of the 8049 to establish a sampling period of four times the desired baud rates. For 2400 baud operation a crystal frequency of 9.216 MHz was chosen after the following calculation:

$$f = 480N(2400)(4)$$

where 480 is the factor by which the crystal frequency is divided within the processor to get the basic interrupt rate

2400 is the desired baud rate

4 is the required number of samples per bit time

N is the value loaded into the MCS-48 timer when it overflows

The value N was chosen to be two (resulting in $f = 9.216$ MHz) so that the operating frequency of the 8049 could be as high as possible without exceeding the maximum frequency specification of the 8049 (11 MHz).

```

;
;
; START OF RECEIVE ROUTINE
; =====
;
;1 IF RECEIVE FLAG=0 THEN
;2   IF SERIAL INPUT=SPACE THEN
;3     RECEIVE FLAG:=1
;3     BYTE FINISHED FLAG:=0
;2   ENDIF
;1 ELSE   SINCE RECEIVE FLAG=1 THEN
;2   IF SYNC FLAG=0 THEN
;3     IF SERIAL INPUT=SPACE THEN
;4       SYNC FLAG:=1
;4       DATA:=80H
;4       SAMPLE CNTR:=4
;3     ELSE   SINCE SERIAL INPUT=MARK THEN
;4       RECEIVE FLAG:=0
;3     ENDIF
;2   ELSE   SINCE SYNC FLAG=1 THEN
;3     SAMPLE COUNTER:=SAMPLE COUNTER-1
;3     IF SAMPLE COUNTER=0 THEN
;4       SAMPLE COUNTER:=4
;4       IF BYTE FINISHED FLAG=0 THEN
;5         CARRY:=SERIAL INPUT
;5         SHIFT DATA RIGHT WITH CARRY
;5         IF CARRY=1 THEN
;6           OKDATA:=DATA
;6           IF DATA READY FLAG=0 THEN
;7             BYTE FINISHED FLAG=1
;6           ELSE
;7             BYTE FINISHED FLAG:=1
;7             OVERRUN FLAG:=1
;6           ENDIF
;5         ENDIF
;4       ELSE   SINCE BYTE FINISHED FLAG=1 THEN
;5         IF SERIAL INPUT=MARK THEN
;6           DATA READY FLAG:=1
;5         ELSE   SINCE SERIAL INPUT=SPACE THEN
;6           ERROR FLAG:=1
;5         ENDIF
;5         RECEIVE FLAG:=0
;5         SYNC FLAG:=0
;4       ENDIF
;3     ENDIF
;2   ENDIF
;1 ENDIF

```

Figure 2

The timer interrupt service routine always loads the timer with a constant value. In effect the timer is used to generate an independent time base of four times the required baud rate. This time base is free running and is never modified by either the receive or transmit programs, thus allowing both of them to use the same timer. Routines which do other time dependent tasks (such as scanning keyboards) can also be called periodically at some fixed multiple of this basic time unit.

The algorithm shown in Figure 2 uses this basic clock plus a handful of flags to process the serial input data.

Once the meaning of these flags are understood the operation of the algorithm should be clear. The **Receive Flag** is set whenever the program is in the process of receiving a character. The **Sync Flag** is set when the center of the start bit has been checked and found to be a SPACE (if a MARK is detected at this point the receiver process has been triggered by a noise pulse so the program clears the **Receive Flag** and returns to the idle state). When the program detects synchronization it loads the variable **DATA** with 80H and starts sampling the serial line every four counts. As the data is received it is right shifted into variable **DATA**; after eight bits have been received the initial one set into **DATA** will result in a carry out and the program knows that it has received all eight bits. At this point it will transfer all eight bits to the variable **OKDATA** and set the **Byte Finished Flag** so that on the next sample it will test for a valid stop bit instead of shifting in data. If this test is successful the **Data Ready Flag** will be set to indicate that the data is available to the main process. If the test is unsuccessful the **Error Flag** will be set.

The transmit algorithm is shown in Figure 3. It is executed immediately following the receive process. It is a simple program which divides the free running clock down and transmits a bit every fourth clock. The variable **TICK COUNTER** is used to do the division. The **Transmitting Flag** indicates when a character transmission is in progress and is also used to determine when the START bit should be sent. The **TICK COUNTER** is used to determine when to send the next bit ($TICK COUNTER \text{ MOD } 4 = 0$) and also when the STOP bits should be sent ($TICK COUNTER = 9 \text{ } 4$). After the transmit routine completes any other timer based routines, such as a keyboard/display scanner or a real time clock, can be executed.

```

;
; START OF TRANSMIT ROUTINE
; =====
;
;1
;1 TICK COUNTER:=TICK COUNTER+1
;1 IF TICK COUNTER MOD 4=0 THEN
;2   IF TRANSMITTING FLAG=1 THEN
;3     IF TICK COUNTER=00 1010 00 BINARY THEN
;4       TRANSMITTING FLAG:=0
;3     ELSE   IF TICK COUNTER=00 1001 00 BINARY THEN
;4       SEND END MARK
;4       TRANSMITTING FLAG:=0
;3     ELSE   SINCE TICK COUNTER>THE ABOVE COUNT THEN
;4       SEND NEXT BIT
;3     ENDIF
;2   ELSE   SINCE TRANSMITTING FLAG=0 THEN
;3     IF TRANSMIT REQUEST FLAG=1 THEN
;4       XMTBYT:=XMTBYT
;4       TRANSMIT REQUEST FLAG:=0
;4       TRANSMITTING FLAG:=1
;4       TICK COUNTER:=0
;4       SEND SYNC BIT (SPACE)
;3     ENDIF
;2   ENDIF
;1 ENDIF

```

Figure 3

Figure 4 shows the complete receive and transmit programs as they are implemented in the instruction set of

the 8049. Also included in Fig. 4 is a short routine which was used to test the algorithm.

```

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

LOC OBJ      SEQ      SOURCE STATEMENT

1 ;*****
2 ;*
3 ;*      THIS PROGRAM TESTS THE FULL DUPLEX COMMUNICATION SOFTWARE
4 ;*
5 ;*****
6 ;
7 $INCLUDE(:F1:URTEST.PDL)
8 ;
9 ;      START OF TEST ROUTINE
10 ;      =====
11 ;
12 ;
13 ;
14 ;
15 ;
16 ; 1 ERROR COUNT:=0
17 ; 1 REPEAT
18 ; 2 PATTERN:=0
19 ; 2 INITIALIZE TIMER
20 ; 2 CLEAR FLAGBYTE
21 ; 2 FLAG1=MARK
22 ; 2 REPEAT
23 ; 3 IF TRANSMIT REQUEST FLAG=0 THEN
24 ; 4     NXTBYTE:=PATTERN
25 ; 4     TRANSMIT REQUEST FLAG=1
26 ; 3     ENDIF
27 ; 3 IF DATA READY FLAG=1 THEN
28 ; 4     PATTERN:=OKDATA
29 ; 4     DATA READY FLAG:=0
30 ; 3     ENDIF
31 ; 2 UNTIL ERROR FLAG OR OVERRUN FLAG
32 ; 2 INCREMENT ERROR COUNT
33 ; 1 UNTIL FOREVER
34 ; EOF
35 $EJECT

0000      36      ORG      0
0000 C5    37 ; 1 SELECT REGISTER BANK 0
0000      38      SEL      RB0
0001 2400  39 ; 1 GOTO TEST
0001      40      JMP      TEST
41 $      INCLUDE(:F1:UART)
42 ;
43 ;
44 ;      ASYNCHRONOUS RECEIVE/TRANSMIT ROUTINE
45 ;      =====
46 ;      THIS ROUTINE RECEIVES SERIAL CODE USING PIN 0 AS RXD
47 ;      AND CONCURRENTLY TRANSMITS USING PIN P27
48 ;      NOTE:
49 ;      THIS ROUTINE USES FLAG 1 TO BUFFER THE TRANSMITTED

```

Figure 4

LOC	OBJ	SEQ	SOURCE STATEMENT	LOC	OBJ
			= 50 ; 1 DATA LINE. THIS ELIMINATES THE JITTER THAT		
			= 51 ; 1 WOULD BE CAUSED BY VARIATIONS IN THE RECEIVE		
			= 52 ; 1 TIMING. NO OTHER PROGRAM MAY USE FLAG 1 WHILE		
			= 53 ; 1 THE TIMER INTERRUPT IS ENABLED.		
			= 54 ; 1 USED TO GENERATE A		
			= 55 ; 1 USED TO GENERATE A		
			= 56 ; 1 USED TO GENERATE A		
			= 57 ; 1 USED TO GENERATE A		
			= 58 ; 1 USED TO GENERATE A		
			= 59 ; REGISTER ASSIGNMENTS-BANK1		
			= 60 ; =====		
			= 61 ;		
			= 62 ;		
0007			= 63 ATEMP EQU R7 ; USED TO SAVE ACCUMULATOR CONTENTS DURING INTERRUPT		
0006			= 64 FLGBYT EQU R6 ; CONTAINS VARIOUS FLAGS USED TO CONTROL THE RECEIVE		
			= 65 ; AND TRANSMIT PROCESS. SEE CONSTANT DEFINITIONS FOR		
			= 66 ; THE MEANING OF EACH BIT		
0005			= 67 SAMCTR EQU R5 ; SAMPLE COUNTER FOR THE RECIEVE PROCESS		
0004			= 68 TCKCTR EQU R4 ; SAMPLE COUNTER FOR THE TRANSMIT PROCESS		
0000			= 69 REG0 EQU R0 ; USED AS POINTER REGISTER		
			= 70 ;		
			= 71 ; RAM ASSIGNMENTS		
			= 72 ; =====		
			= 73 ;		
0020			= 74 MOKDAT EQU 20H ; RECEIVE RETURNS VALID DATA IN THIS BYTE		
0021			= 75 MDATA EQU 21H ; RECEIVE ACCUMULATES DATA IN THIS BYTE		
0022			= 76 MXMTBY EQU 22H ; CONTAINS BYTE BEING TRANSMITTED		
0023			= 77 MNXTBY EQU 23H ; CONTAINS THE NEXT BYTE TO BE TRANSMITTED		
			= 78 \$EJECT		
			= 79 ;		
			= 80 ;		
			= 81 ; CONSTANTS		
			= 82 ; =====		
			= 83 ;		
			= 84 ; THE FOLLOWING CONSTANTS ARE USED TO ACCESS THE FLAG BITS CONTAINED		
			= 85 ; IN REGISTER FLGBYT		
			= 86 ;		
0001			= 87 RCVFLG EQU 01H ; SET WHEN START BIT IS FIRST DETECTED		
			= 88 ; RESET WHEN RECEIVE PROCESS IS COMPLETE		
0002			= 89 SYNFLG EQU 02H ; SET WHEN START BIT IS VERIFIED		
			= 90 ; RESET WHEN RECEIVE PROCESS IS COMPLETE		
0004			= 91 BYFNFL EQU 04H ; RESET WHEN START BIT IS FIRST DETECTED		
			= 92 ; SET WHEN THE EIGHT DATA BITS HAVE ALL BEEN RECEIVED		
0008			= 93 DRDYFL EQU 08H ; SHOULD BE RESET BY MAIN PROGRAM WHEN DATA IS ACCEPTED		
			= 94 ; SET BY RECEIVE PROCESS WHEN STOP BIT(S) ARE VERIFIED		
0010			= 95 ERRFLG EQU 10H ; SHOULD BE RESET BY MAIN PROGRAM WHEN SAMPLED		
			= 96 ; SET BY RECEIVE PROCESS IF A FRAMING ERROR IS DETECTED		
0020			= 97 TRROFL EQU 20H ; TESTED BY MAIN PROGRAM TO DETERMINE IF READY TO		
			= 98 ; TRANSMIT A NEW BYTE-SET TO INDICATE THAT NXTBYT		
			= 99 ; HAS BEEN LOADED		
			= 100 ; RESET BY TRANSMIT PROCESS WHEN BYTE IS ACCEPTED		
0040			= 101 TRNGFL EQU 40H ; SET WHEN TRANSMISSION OF A BYTE STARTS		
			= 102 ; RESET WHEN STOP BIT IS TRANSMITTED		
0080			= 103 OVRUN EQU 80H ; SET BY RECEIVE PROCESS WHEN OVERUN OCCURS		
			= 104 ; SHOULD BE RESET BY MAIN PROGRAM WHEN SAMPLED		

Figure 4 (continued)

```

= 107 ; *****
= 108 ;
0080 = 109 MARK EQU 80H ; USED TO GENERATED A MARK
FF7F = 110 SPACE EQU NOT 80H ; USED TO GENERATE A SPACE
0000 = 111 STPBTS EQU 0 ; CONTROLS THE NUMBER OF STOP BITS
= 112 ; 0 GENERATES ONE STOP BIT
= 113 ; 1 GENERATES TWO STOP BITS
= 114 ;
= 115 $EJECT *****
= 116 ;
= 117 ; START OF RECEIVE/TRANSMIT INTERRUPT SERVICE ROUTINE
*****
0007 = 118 ; *****
= 119 ;
0007 = 120 ; ORG 0007H
= 121 ;
= 122 ; 1 ENTER INTERRUPT MODE
0007 160A = 123 TISR:JTF UART
0009 93 = 124 RETR
000A D5 = 125 UART: SEL RB1
= 126 ; 1 SAVE ACCUMULATOR CONTENTS
000B AF = 127 MOV ATEMP,A
= 128 ; 1 RELOAD TIMER
= 129 MOV A:#TIMCNT
000C 23FE = 130 MOV T,A
000E 62 = 131
= 132 ; OUTPUT TXD BUFFER (F1) TO TXD I/O LINE (P27)
= 133 ;
= 134 ;
000F 7615 = 135 JF1 OMARK
0011 9A7F = 136 OSPACE: ANL P2,#SPACE
0013 0417 = 137 JMP RCV000
0015 8A80 = 138 OMARK: ORL P2,#MARK
= 139 ;
= 140 ; START OF RECEIVE ROUTINE
= 141 ; *****
= 142 ;
= 143 ; 1 IF RECEIVE FLAG=0 THEN
0017 FE = 144 RCV000: MOV A,FLGBYT
0018 1224 = 145 JB0 RCV010
= 146 ; 2 IF SERIAL INPUT=SPACE THEN
001A 3664 = 147 JT0 XMIT
= 148 ; 3 RECEIVE FLAG:=1
001C FE = 149 MOV A,FLGBYT
001D 4301 = 150 ORL A,#RCVFLG
= 151 ; 3 BYTE FINISHED FLAG:=0
001F 53FB = 152 ANL A,#NOT BYFNFL
= 153 ; 2 ENDF
0021 AE = 154 MOV FLGBYT,A
0022 0464 = 155 JMP XMIT
= 156 ; 1 ELSE SINCE RECEIVE FLAG=1 THEN
= 157 ; 2 IF SYNC FLAG=0 THEN
0024 3238 = 158 RCV010: JB1 RCV030
= 159 ; 3 IF SERIAL INPUT=SPACE THEN

```

Figure 4 (continued)

LOC	OBJ	SEQ	SOURCE STATEMENT	OBJ2	LOC	OBJ
0026	3633	= 160	RCV020	A: RCV =		
		= 161	SYNC FLAG:=1	B: RCV =		
0028	4302	= 162	ORL #0, A, #SYNFLAG	C: RCV =		
002A	AE	= 163	MOV #0, FLGBYT, A	D: RCV =		
		= 164	DATA:=80H	E: RCV =		
002B	B821	= 165	MOV R0, #MDATA	F: RCV =		
002D	B080	= 166	MOV R0, #00H	G: RCV =		
		= 167	SAMPLE CNTR:=4	H: RCV =		
002F	BD04	= 168	MOV SAMCTR, #4	I: RCV =		
0031	0464	= 169	JMP XMIT	J: RCV =		
		= 170	ELSE SINCE SERIAL INPUT=MARK THEN	K: RCV =		
		= 171	RECEIVE FLAG:=0	L: RCV =		
0033	53FE	= 172	AND #0, A, #NOT RCVFLAG	M: RCV =		
		= 173	ENDIF	N: RCV =		
0035	AE	= 174	MOV FLGBYT, A	O: RCV =		
0036	0464	= 175	JMP XMIT	P: RCV =		
		= 176	ELSE SINCE SYNC FLAG=1 THEN	Q: RCV =		
		= 177	SAMPLE COUNTER:=SAMPLE COUNTER-1	R: RCV =		
0038	ED64	= 178	RCV030: DJNZ SAMCTR, XMIT	S: RCV =		
		= 179	IF SAMPLE COUNTER=0 THEN	T: RCV =		
		= 180	SAMPLE COUNTER:=4	U: RCV =		
003A	BD04	= 181	MOV SAMCTR, #4	V: RCV =		
		= 182	IF BYTE FINISHED FLAG=0 THEN	W: RCV =		
003C	5259	= 183	JB2 RCV050	X: RCV =		
003E	97	= 184	CLR CARRY	Y: RCV =		
		= 185	CARRY:=SERIAL INPUT	Z: RCV =		
003F	2642	= 186	JNZ RCV040	AA: RCV =		
0041	A7	= 187	CPL C	AB: RCV =		
0042	B821	= 188	RCV040: MOV R0, #MDATA	AC: RCV =		
0044	F0	= 189	MOV R0, A, R0	AD: RCV =		
		= 190	SHIFT DATA RIGHT WITH CARRY	AE: RCV =		
0045	67	= 191	RCV050: MOV R0, A, R0	AF: RCV =		
0046	A0	= 192	MOV R0, A, R0	AG: RCV =		
		= 193	IF CARRY=1 THEN	AH: RCV =		
0047	E664	= 194	JNC XMIT	AI: RCV =		
		= 195	DATA:=DATA	AJ: RCV =		
0049	B820	= 196	MOV R0, #MDATA	AK: RCV =		
004B	A0	= 197	MOV R0, A, R0	AL: RCV =		
		= 198	IF DATA READ FLAG=0 THEN	AM: RCV =		
004C	FE	= 199	MOV R0, A, FLGBYT	AN: RCV =		
004D	7254	= 200	JB3 RCV045	AO: RCV =		
		= 201	BYTE FINISHED FLAG=1	AP: RCV =		
004F	4304	= 202	ORL #0, A, #BYFNFL	AQ: RCV =		
0051	AE	= 203	MOV FLGBYT, A	AR: RCV =		
0052	0464	= 204	JMP XMIT	AS: RCV =		
		= 205	ELSE	AT: RCV =		
		= 206	BYTE FINISHED FLAG:=1	AU: RCV =		
		= 207	OVERRUN FLAG:=1	AV: RCV =		
		= 208	RCV045: MOV R0, A, R0	AW: RCV =		
		= 209	MOV R0, A, FLGBYT	AX: RCV =		
0054	4384	= 210	ORL #0, A, #BYFNFL OR OVRUN	AY: RCV =		
0056	AE	= 211	MOV FLGBYT, A	AZ: RCV =		
		= 212	ENDIF	BA: RCV =		
		= 213	ENDIF	BB: RCV =		
0057	0464	= 214	JMP XMIT	BC: RCV =		

Figure 4 (continued)

LOC	OBJ	SEQ	SOURCE STATEMENT	0002	0002	100	001
		= 215 ; 4	ELSE SINCE BYTE FINISHED FLAG=1 THEN				
		= 216 ; 5	IF SERIAL INPUT=MARK THEN				
0059	265F	= 217 RCV050: JNT02 RCV060					
		= 218 ; 6	DATA-READY FLAG:=1				
005B	4308	= 219 ORL R02: A, #DRDYFL					
005D	0461	= 220 JMP R02: RCV070					
		= 221 ; 5	ELSE SINCE SERIAL INPUT=SPACE THEN				
		= 222 ; 6	ERROR FLAG:=1				
005F	4310	= 223 RCV060: ORL R02: A, #ERRFLG					
		= 224 ; 5	ENDIF				
		= 225 ; 5	RECEIVE FLAG:=0				
		= 226 ; 5	SYNC FLAG:=0				
0061	53FC	= 227 RCV070: ANL R02: A, #NOT(SYNFLG OR RCVFLG)					
0063	AE	= 228 MOV FLGBYT, A					
		= 229 ; 4	ENDIF				
		= 230 ; 3	ENDIF				
		= 231 ; 2	ENDIF				
		= 232 ; 1	ENDIF				
		= 233	EJECT				
		= 234	RESTART OF TRANSMIT ROUTINE				
		= 235 ;					
		= 236 ;					
		= 237	IF BYTE FINISHED				
		= 238 ; 1					
		= 239	TRANSMITTER OUTPUT BIT IS P2-7				
		= 240	TICK COUNTER:=TICK COUNTER+1				
0064	1C	= 241 XMT: INC R02: TCKCTR					
		= 242 ; 1	IF TICK COUNTER MOD 4=0 THEN				
0065	2303	= 243 MOV R02: A, #03H					
0067	5C	= 244 ANL R02: A, TCKCTR					
0068	9697	= 245 R02: JNZ T01: RETURN					
		= 246 ; 2	IF TRANSMITTING FLAG=1 THEN				
006A	FE	= 247 MOV R02: A, FLGBYT					
006B	37	= 248 R02: CPL R02: A					
006C	D286	= 249 JB6 T01: XMT040					
		= 250	IF STPBT5 EQ 1				
		= 251 ; 3	IF TICK COUNTER=00 1010 00 BINARY THEN				
		= 252	MOV R02: A, #28H				
		= 253	R02: XRL R02: A, TCKCTR				
		= 254	JNZ R02: XMT010				
		= 255 ; 4	TRANSMITTING FLAG:=0				
		= 256	MOV R02: A, FLGBYT				
		= 257	ANL R02: A, #NOT TRNGFL				
		= 258	MOV R02: FLGBYT, A				
		= 259	JMP T01: RETURN				
		= 260	ENDIF				
		= 261 ; 3	ELSE IF TICK COUNTER=00 1001 00 BINARY THEN				
006E	2324	= 262 XMT010: MOV R02: A, #24H					
0070	DC	= 263 XRL R02: A, TCKCTR					
0071	967B	= 264 JNZ R02: XMT020					
		= 265 ; 4	SEND END MARK				
0073	A5	= 266 CLR R02: F1					
0074	B5	= 267 CPL R02: F1					
		= 268	IF STPBT5 EQ 0				
		= 269 ; 4	TRANSMITTING FLAG:=0				

Figure 4 (continued)

LOC	OBJ	SEQ	SOURCE STATEMENT	LOC	OBJ
0075	FE	= 270	MOV A, FLGBYT ; CONDITIONAL ASSEMBLY		
0076	53BF	= 271	ANL A, #NOT TRNGFL ;		
0078	AE	= 272	MOV FLGBYT, A ;		
0079	0497	= 273	JMP RETURN ;		
		= 274	ENDIF		
		= 275 ; 3	ELSE SINCE TICK COUNTER > THE ABOVE COUNT THEN		
		= 276 ; 4	SEND NEXT BIT		
007B	B822	= 277	XMT020: MOV R0, #XMTBYT		
007D	F0	= 278	MOV A, @R0		
007E	67	= 279	RRC A		
007F	A0	= 280	MOV @R0, A		
0080	A5	= 281	CLR F1 ; FLAG 1 WILL BE USED TO BUFFER TXD		
0081	E697	= 282	RETURN ; GO TO RETURN POINT IF TXD=SPACE (0)		
0083	B5	= 283	CPL F1 ; ELSE COMPLEMENT FLAG 1 TO A MARK		
0084	0497	= 284	JMP RETURN		
		= 285 ; 3	ENDIF		
		= 286 ; 2	ELSE SINCE TRANSMITTING FLAG=0 THEN		
		= 287 ; 3	IF TRANSMIT REQUEST FLAG=1 THEN		
0086	B297	= 288	XMT040: JBS RETURN ; FLAG BYTE THERE		
		= 289 ; 4	XMTBYT:=NXTBYT		
0088	B823	= 290	MOV R0, #NXTBYT		
008A	F0	= 291	MOV A, @R0		
008B	B822	= 292	MOV R0, #XMTBYT		
008D	A0	= 293	MOV @R0, A		
		= 294 ; 4	TRANSMIT REQUEST FLAG:=0		
008E	FE	= 295	MOV A, FLGBYT		
008F	53DF	= 296	ANL A, #NOT TRNGFL		
		= 297 ; 4	TRANSMITTING FLAG:=1		
0091	4340	= 298	ORL A, #TRNGFL		
0093	AE	= 299	MOV FLGBYT, A		
		= 300 ; 4	TICK COUNTER:=0		
0094	BC00	= 301	MOV TCKCTR, #0		
		= 302 ; 4	SEND SYNC BIT (SPACE)		
0096	A5	= 303	CLR F1 ; SET FLAG 1 TO CAUSE A SPACE		
		= 304 ; 3	ENDIF		
		= 305 ; 2	ENDIF		
		= 306 ; 1	ENDIF		
		= 307	RETURN:		
		= 308 ; 1	RESTORE ACCUMULATOR		
0097	FF	= 309	MOV A, ATEMP		
0098	93	= 310	RETR		
		311	\$EJECT		
		312 ;			
		313 ;	START OF TEST ROUTINE		
		314 ;	=====		
		315 ;			
0100		316	ORG 0100H		
FFFE		317	TIMCNT EQU -2		
001E		318	NFLGBY EQU 1EH		
001D		319	MSAMCT EQU 1DH		
001C		320	MTCKCT EQU 1CH		
		321 ;			
0007		322	ERRCNT EQU R7		
0006		323	PATT EQU R6		
		324 ;			

Figure 4 (continued)

LOC	OBJ	SEQ	SOURCE STATEMENT	DEST	LOC
		325 ;			
		326 ;			
		327 ; 1 ERROR COUNT:=0			
0100	BF00	328 TEST: MOV	ERRCNT, #0		
		329 ; 1 REPEAT			
		330 TLOP:			
		331 ; 2 PATTERN:=0			
0102	BE00	332	MOV PATT, #00		
		333 ; 2	INITIALIZE TIMER		
0104	23FE	334	MOV A, #TIMCNT		
0106	62	335	MOV T, A		
0107	55	336	STRT T		
0108	25	337	EN TCNTI		
		338 ; 2	CLEAR FLAGBYTE		
0109	B81E	339	MOV R0, #HFLGBY		
010B	B000	340	MOV @R0, #0		
		341 ; 2	FLAG1=MARK		
010D	A5	342	CLR F1		
010E	B5	343	CPL F1		
		344 ; 2	REPEAT		
		345 TILOP:			
		346 ; 3	IF TRANSMIT REQUEST FLAG=0 THEN		
010F	B81E	347	MOV R0, #HFLGBY		
0111	F0	348	MOV A, @R0		
0112	B224	349	JB5 TREC		
		350 ; 4	NXTBYTE:=PATTERN		
0114	B923	351	MOV R1, #NXTBY		
0116	FE	352	MOV A, PATT		
0117	A1	353	MOV @R1, A		
		354 ; 4	TRANSMIT REQUEST FLAG=1		
0118	35	355	DIS TCNTI ; LOCK OUT TIMER INTERRUPT		
		356	SO THAT MUTUAL EXCLUSION IS MAINTAINED WHILE		
		357	THE FLAG BYTE IS BEING MODIFIED		
0119	F0	358	MOV A, @R0		
011A	4320	359	ORL A, #TRROFL		
011C	A0	360	MOV @R0, A		
011D	25	361	EN TCNTI		
011E	1622	362	JTF TESTA		
0120	2424	363	JMP TREC		
0122	140A	364 TESTA: CALL	UART ; CALL UART BECAUSE TIMER OVERFLOWED DURING LOCKOUT		
		365 ; 3	ENDIF		
		366 ; 3	IF DATA READY FLAG=1 THEN		
		367 TREC:			
0124	F0	368	MOV A, @R0		
0125	37	369	CPL A		
0126	7238	370	JB3 TRECE		
		371 ; 4	PATTERN:=OKDATA		
0128	B920	372	MOV R1, #OKDAT		
012A	F1	373	MOV A, @R1		
012B	AE	374	MOV PATT, A		
		375 ; 4	DATA READY FLAG:=0		
012C	35	376	DIS TCNTI ; LOCK OUT TIMER INTERRUPT		
		377	SO THAT MUTUAL EXCLUSION IS MAINTAINED WHILE		
		378	THE FLAG BYTE IS BEING MODIFIED		
012D	F0	379	MOV A, @R0		

Figure 4 (continued)

```

012E 53F7 380 ANL A,#NOT DRDYFL
0130 A0 381 MOV @R0,A
0131 25 382 EN TCNTI
0132 1636 383 JTF TESTB
0134 2438 384 JMP TRECE
0136 140A 385 TESTB: CALL UART ; CALL UART IF TIMER OVERFLOWED DURING LOCKOUT
0138 F0 386 TRECE: ;
0139 5390 387 ;3 ENDF
013B C60F 388 ;2 UNTIL ERROR FLAG OR OVERRUN FLAG
013D 1F 389 MOV A,@R0
013E 2402 390 ANL A,#(OVRUN OR ERRFLG)
013F 391 JZ TILOP
0140 392 ;2 INCREMENT ERROR COUNT
0141 393 INC ERRCNT
0142 394 ;1 UNTIL FOREVER
0143 2402 395 JMP TLOP
0144 396 ;EOF
0145 397 END

```

USER SYMBOLS

```

ATEMP 0007 BYFNFL 0004 DRDYFL 0008 ERRCNT 0007
MFLGBY 001E MNXTBY 0023 MOKDAT 0020 MSAMCT 0010
OVRUN 0000 PATT 0006 RCV000 0017 RCV010 0024
RCV050 0059 RCV060 005F RCV070 0061 RCVFLG 0001
STPBTS 0000 SYNFLG 0002 TCKCTR 0004 TEST 0100
TISR 0007 TLOP 0102 TRECE 0124 TRECE 0138
XMT010 006E XMT020 007B XMT040 0086

```

ASSEMBLY COMPLETE, NO ERRORS

Figure 4 (continued)

All mnemonics copyrighted © Intel Corporation 1979.

MULTIPLY ALGORITHMS

Most microcomputer programmers have at one time or another implemented a multiply routine as part of a larger program. The usual procedure is to find an algorithm that works and modify it to work on the machine being used. There is nothing wrong with this approach. If engineers felt that they had to reinvent the wheel every time a new design is undertaken, that's probably what most of us would be doing—designing wheels. If the efficiency of the multiply algorithm, either in terms

The sum of all these terms represents the product of A and B. The simplest multiply algorithm factors the above terms as follows:

$$A \times B = B0(A) \cdot 2^0 + B1(A) \cdot 2^1 + B2(A) \cdot 2^2 + B3(A) \cdot 2^3$$

Since the coefficients of B (i.e., B0, B1, B2, and B3) can only take on the binary values of 1 or 0, the sum of the products can be formed by two. The simplest implementation of this would be:

```

MULTIPLY:
    PRODUCT=0
    IF B0=1 THEN PRODUCT=PRODUCT+A
    IF B1=1 THEN PRODUCT=PRODUCT+A*2
    IF B2=1 THEN PRODUCT=PRODUCT+A*4
    IF B3=1 THEN PRODUCT=PRODUCT+A*8
END MULTIPLY

```

In order to conserve memory, the above straight line code is normally converted to the following loop:

```

MULTIPLY:
    PRODUCT=0
    COUNT=4
    UNTIL COUNT=0
    IF B=1 THEN PRODUCT=PRODUCT+A
    COUNT=COUNT-1
    A=A*2
END MULTIPLY

```

The repeated multiplication of A by two (which can be performed by a simple left shift) forms the terms 2^0A , 2^1A , and 2^2A . The variable B is divided by two (performed by a simple right shift) so that the least significant bit can always be used to determine whether addition should be executed during each pass through the loop. It is from these shifting and addition operations that the following algorithm is derived:

of code size or execution time is important, however, it is necessary to be reasonably familiar with the multiplication process so that appropriate optimizations for the machine being used can be made.

To understand how multiplication operates in the binary number system, consider the multiplication of two four bit operands A and B. The "ones and zeros" in A and B represent the coefficients of two polynomials. The operation $A \times B$ can be represented as the following multiplication of polynomials:

$$\begin{array}{r}
 \begin{array}{ccccccc}
 & A^3 \cdot 2^3 & + & A^2 \cdot 2^2 & + & A^1 \cdot 2^1 & + & A^0 \cdot 2^0 \\
 \times & B^3 \cdot 2^3 & + & B^2 \cdot 2^2 & + & B^1 \cdot 2^1 & + & B^0 \cdot 2^0 \\
 \hline
 & & & & & B^0A^3 \cdot 2^3 & + & B^0A^2 \cdot 2^2 & + & B^0A^1 \cdot 2^1 & + & B^0A^0 \cdot 2^0 \\
 & & & & B^1A^3 \cdot 2^4 & + & B^1A^2 \cdot 2^3 & + & B^1A^1 \cdot 2^2 & + & B^1A^0 \cdot 2^1 \\
 & & B^2A^3 \cdot 2^5 & + & B^2A^2 \cdot 2^4 & + & B^2A^1 \cdot 2^3 & + & B^2A^0 \cdot 2^2 \\
 + & B^3A^3 \cdot 2^6 & + & B^3A^2 \cdot 2^5 & + & B^3A^1 \cdot 2^4 & + & B^3A^0 \cdot 2^3
 \end{array}
 \end{array}$$

The sum of all these terms is the product of A and B. Since the coefficients of B (i.e., B0, B1, B2, and B3) can only take on the binary values of 1 or 0, the sum of the products can be formed by a series of simple adds and multiplications by two. The simplest implementation of this would be:

```
MULTIPLY:
  PRODUCT = 0
  IF B0 = 1 THEN PRODUCT = PRODUCT + A
  IF B1 = 1 THEN PRODUCT = PRODUCT + 2*A
  IF B2 = 1 THEN PRODUCT = PRODUCT + 4*A
  IF B3 = 1 THEN PRODUCT = PRODUCT + 8*A
END MULTIPLY
```

In order to conserve memory, the above straight line code is normally converted to the following loop:

```
MULTIPLY:
  PRODUCT = 0
  COUNT = 4
  REPEAT
    IF B[0] = 1 THEN PRODUCT = PRODUCT + A
    A = 2*A
    B = B/2
    COUNT = COUNT - 1
  UNTIL COUNT = 0
END MULTIPLY
```

The repeated multiplication of A by two (which can be performed by a simple left shift) forms the terms 2*A, 4*A, and 8*A. The variable B is divided by two (performed by a simple right shift) so that the least significant bit can always be used to determine whether the addition should be executed during each pass through the loop. It is from these shifting and addition opera-

the partial product is double precision relative to the two operands. The other problem, which is also related to double precision operations, is that the A operand is double precision and that it must be left shifted and then the B operand must be right shifted. An examination of the "longhand" polynomial multiplication will reveal that, although the partial product is indeed double precision, each addition performed is only single precision. It would be desirable to be able to shift the partial product as it is formed so that only single precision additions are performed. This would be especially true if the partial product could be shifted into the "B" operand since one bit of the partial product is formed during each pass through the loop and (happily) one bit of the "B" operand is vacated. To do this, however, it is necessary to modify the algorithm so that both of the shifts that occur are of the same type.

To see how this can be done one can take the basic multiplication equation already presented:

$$A * B = B0 * (A * 2^0) + B1 * (A * 2^1) + B2 * (A * 2^2) + B3 * (A * 2^3)$$

and factoring 2^4 from the right side:

$$A * B = 2^4 [B0 * (A * 2^{-4}) + B1 * (A * 2^{-3}) + B2 * (A * 2^{-2}) + B3 * (A * 2^{-1})]$$

This operation has resulted in a term (within the brackets) which can be formed by right shifts and adds and then multiplied by 2^4 to get the final result. The resulting algorithm, expanded to form an eight by eight multiplication, is shown in figure 5. Note that although the result is a full sixteen bits, the algorithm only performs eight bit additions and that only a single sixteen bit shift operation is involved. This has the effect of reducing both the code space and the execution time for the routine.

IS15-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

LOC	OBJ	SEQ	SOURCE STATEMENT
		1	\$MACROFILE
		2	\$INCLUDE(:F1:MPY8.HED)
		3	*****
		4	;
		5	;
		6	;
		7	*****
		8	;
		9	;
		10	THIS UTILITY PROVIDES AN 8 BY 8 UNSIGNED MULTIPLY
		11	AT ENTRY:
		12	A = LOWER EIGHT BITS OF DESTINATION OPERAND
		13	XA = DON'T CARE
		14	R1 = POINTER TO SOURCE OPERAND (MULTIPLIER) IN INTERNAL MEMORY

Figure 5

LOC	OBJ	SEQ	SOURCE STATEMENT	OBJECT CODE	LOC
		= 68	MPY8A:		
		= 69	:3		
			MULTPLICAND:=MULTPLICAND/2		
*	000E 2A	= 70	XCH A,XA		
*	000F 61	= 71	ADD T,A,0010 2718		
*	0010 67	= 72	RRC A		
*	0011 2A	= 73	XCH A,XA		
*	0012 67	= 74	RRC A		
	0013 E004	= 75	DJNZ COUNT,MPY8LP		
	0015 83	= 76	RET		
		= 77	:3		
			MULTPLICAND:=MULTPLICAND/2		
		= 78	:2		
			ENDIF		
		= 79	:2		
			COUNT:=COUNT-1		
		= 80	:1		
			UNTIL COUNT=0		
		= 81	:1		
			END MPY8X8		
			END		

USER SYMBOLS

NAME	LOC	OBJ	SEQ	SOURCE STATEMENT	OBJECT CODE	LOC
COUNT	0003	DIGPR	0003	ICNT 0004	MPY8A 000E	MPY8LP 0004
MPY8X8	0000					
XA	0002					

ASSEMBLY COMPLETE, NO ERRORS

All mnemonics copyrighted © Intel Corporation 1979.

DIVIDE ALGORITHMS

In order to understand binary division a four bit operation will again be used as an example. The following algorithm will perform a four by four division:

DIVIDE:

```

IF 16*DIVISOR>= DIVIDEND THEN
  SET OVERFLOW ERROR FLAG
ELSE
  IF 8*DIVISOR>= DIVIDEND THEN
    QUOTIENT[3]=1
    DIVIDEND:=DIVIDEND-8*DIVISOR
  ELSE
    QUOTIENT[3]=0
  ENDIF
  IF 4*DIVISOR>= DIVIDEND THEN
    QUOTIENT[2]=1
    DIVIDEND:=DIVIDEND-4*DIVISOR
  ELSE
    QUOTIENT[2]=0
  ENDIF
  IF 2*DIVISOR>= DIVIDEND THEN
    QUOTIENT[1]=1
    DIVIDEND:=DIVIDEND-2*DIVISOR
  ELSE
    QUOTIENT[1]=0
  ENDIF
  IF 1*DIVISOR>= DIVIDEND THEN
    QUOTIENT[0]=1
    DIVIDEND:=DIVIDEND-1*DIVISOR
  ELSE
    QUOTIENT[0]=0
  ENDIF
ENDIF
END DIVIDE

```

The algorithm is easy to understand. The first test asks if the division will fit into the dividend sixteen times. If it will, the quotient cannot be expressed in only four bits so an overflow error flag is set and the divide algorithm ends. The algorithm then proceeds to determine if eight times the divisor fits, four times, etc. After each test it either sets or clears the appropriate quotient bit and modifies the dividend. To see this algorithm in action, consider the division of 15 by 5:

00001111	(15)	
- 01010000	(16*5)	Doesn't fit—no overflow
00001111	(15)	
- 00101000	(8*5)	Doesn't fit—Q[3]=0
00001111	(15)	
- 00010100	(4*5)	Doesn't fit—Q[2]=0
00001111	(15)	
- 00001010	(2*5)	Fits—Q[1]=1
00000101	(15-2*5)	
- 00000101	(1*5)	Fits—Q[0]=1
00000000		

The result is Q=0011 which is the binary equivalent of 3—the correct answer. Clearly this algorithm can (and has been) converted to a loop and used to perform divisions. An examination of the procedure, however, will show that it has the same problems as the original multiplication algorithm.

The first problem is that double precision operations are involved with both the comparison of the division with the dividend and the conditional subtraction. The second problem is that as the quotient bits are derived they must be shifted into a register. In order to reduce the register requirements, it would be desirable to shift them into the divisor register as they are generated since the divisor register gets shifted anyway. Unfortunately the quotient bits are derived most significant bits first so doing this will form a mirror image of the quotient—not very useful.

Both of these problems can be solved by observing that the algorithm presented for divide will still work if both sides of all the “equations” involving the dividend are divided by sixteen. The looping algorithm then would proceed as follows:

```
DIVIDE:
QUOTIENT:= 0
COUNT:= 4
DIVIDEND:= DIVIDEND/16
IF DIVISOR>= DIVIDEND THEN
  OVERFLOW FLAG:= 1
ELSE
  REPEAT
    DIVIDEND:= DIVIDEND*2
    QUOTIENT:= QUOTIENT*2
    IF DIVISOR>= DIVIDEND THEN
      QUOTIENT:= QUOTIENT + 1/16 SET QUOTIENT[0]*/
      DIVIDEND:= DIVIDEND - DIVISOR
    ENDIF
    COUNT:= COUNT - 1
  UNTIL COUNT= 0
ENDIF
END DIVIDE
```

When this algorithm is implemented on a computer which does not have a direct compare instruction the comparison is done by subtraction and the inner loop of the algorithm is modified as follows:

```
REPEAT
  DIVIDEND:= DIVIDEND*2
  QUOTIENT:= QUOTIENT*2
  DIVIDEND:= DIVIDEND - DIVISOR
  IF BORROW= 0 THEN
    QUOTIENT:= QUOTIENT + 1
  ELSE
    DIVIDEND:= DIVIDEND + DIVISOR
  ENDIF
  COUNT:= COUNT - 1
UNTIL COUNT= 0
```

An implementation of this algorithm using the 8049 instruction set is shown in figure 6. This routine does an unsigned divide of a 16 bit quantity by an eight bit quantity. Since the multiply algorithm of figure 5 generates a 16 bit result from the multiplication of two eight bit operands, these two routines complement each other and can be used as part of more complex computations.

IS15-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

LOC	OBJ	SEQ	SOURCE STATEMENT
		1	\$MACROFILE
		2	\$INCLUDE(:F1:DIV16.HED)
		3	*****
		4	;
		5	;
		6	;
		7	*****
		8	;
		9	;
		10	;
		11	;
		12	;
		13	;
		14	;
		15	;
		16	;
		17	;

Figure 6

```

20 ; *****
21 ;
22 ;
23 $INCLUDE(:F1:DIV16.PDL)
24 ;1 DIV16:
25 ;1 COUNT:=8
26 ;1 DIVIDEND[15-8]:=DIVIDEND[15-8]-DIVISOR
27 ;1 IF BORROW=0 THEN /* IT FITS*/
28 ;2 SET OVERFLOW FLAG
29 ;1 ELSE
30 ;2 RESTORE DIVIDEND
31 ;2 REPEAT
32 ;3 DIVIDEND:=DIVIDEND*2
33 ;3 QUOTIENT:=QUOTIENT*2
34 ;3 DIVIDEND[15-8]:=DIVIDEND[15-8]-DIVISOR
35 ;3 IF BORROW=1 THEN
36 ;4 RESTORE DIVIDEND
37 ;3 ELSE
38 ;4 QUOTIENT[0]:=1
39 ;3 ENDIF
40 ;3 COUNT:=COUNT-1
41 ;2 UNTIL COUNT=0
42 ;2 CLEAR OVERFLOW FLAG
43 ;1 ENDIF
44 ;1 ENDDIVIDE
45 ; EQUATES
46 ; =====
47 ;
48 ;
0002 49 XA EQU R2
0003 50 COUNT EQU R3
51 ;
52 $EJECT
53 $INCLUDE(:F1:DIV16)
54 ;1 DIV16:
0000 2A 55 DIV16: XCH A,XA ; ROUTINE WORKS MOSTLY WITH BITS 15-8
56 ;1 COUNT:=8
0001 B808 57 MOV COUNT,#8
58 ;1 DIVIDEND[15-8]:=DIVIDEND[15-8]-DIVISOR
0003 37 59 CPL A
0004 61 60 ADD A,@R1
0005 37 61 CPL A
62 ;1 IF BORROW=0 THEN /* IT FITS*/
0006 F608 63 JC DIVIA
64 ;2 SET OVERFLOW FLAG
0008 A7 65 CPL C
0009 0424 66 JMP DIVIB
67 ;1 ELSE
68 DIVIA:
69 ;2 RESTORE DIVIDEND
000B 61 70 ADD A,@R1
71 ;2 REPEAT
72 DIVILP:
73 ;3 DIVIDEND:=DIVIDEND*2

```

Figure 6 (continued)

```

0000 2A      = 76      XCH      A, XA
000E F7      = 77      RLC      A
000F 2A      = 78      XCH      A, XA
0010 F7      = 79      RLC      A
0011 E618    = 80      JNC      DIVIE
0013 37      = 81      CPL      A
0014 61      = 82      ADD      A, @R1
0015 37      = 83      CPL      A
0016 0420    = 84      JMP      DIVIC
          = 85 ; 038 00DIVIDEND[15-8]:=DIVIDEND[15-8]-DIVISOR
0018 37      = 86 DIVIE: CPL      A
0019 61      = 87      ADD      A, @R1
001A 37      = 88      CPL      A
          = 89 ; 3 IF BORROW=1 THEN
001B E620    = 90      JNC      DIVIC
          = 91 ; 4      RESTORE DIVIDEND
001D 61      = 92      ADD      A, @R1
001E 0421    = 93      JMP      DIVID
          = 94 ; 3      ELSE
          = 95 DIVIC:
          = 96 ; 4      QUOTIENT[0]:=1
0020 1A      = 97      INC      XA
          = 98 ; 3      ENDIF
          = 99 ; 3      COUNT:=COUNT-1
          = 100 ; 2 UNTIL COUNT=0
0021 EB0C    = 101 DIVID: DJNZ     COUNT, DIVILP
          = 102 ; 2 CLEAR OVERFLOW FLAG
0023 97      = 103      CLR      C
          = 104 ; 1 ENDIF
          = 105 ; 1 ENDDIVIDE
0024 2A      = 106 DIVIB: XCH      A, XA
0025 83      = 107      RET
          108 END

USER SYMBOLS
COUNT 0003  DIV16 0000  DIV1A 000B  DIV1B 0024  DIVIC 0020  DIVID 0021  DIVIE 0018  DIVILP 000C
XA      0002

ASSEMBLY COMPLETE, NO ERRORS

```

Figure 6 (continued)

All mnemonics copyrighted © Intel Corporation 1979.

BINARY AND BCD CONVERSIONS

The conversion of a binary value to a BCD (binary coded decimal) number can be done with a very straightforward algorithm:

```

CONVERT_TO_BCD:
    BCDACCUM:=0
    COUNT:=PRECISION
    REPEAT
        BIN:=BIN * 2
        BCD:=BCD * 2 + CARRY
        COUNT:=COUNT-1
    UNTIL COUNT=0
    END CONVERT_TO_BCD

```

The variable **BCDACCUM** is a BCD string used to accumulate the result; the variable **BIN** is the binary number to be converted. **PRECISION** is a constant which gives the length, in binary bits of BIN. To see how this works, assume that BIN is a sixteen bit value with the most significant bit set. On the first pass through the loop the multiplication of **BIN** will result in a carry and this carry will be added to BCD. On the remaining passes through the loop BCD will be multiplied by two 15 times. The initial carry into BCD will be multiplied by 2^{15} or 32678, which is the "value" of the most significant bit of **BIN**. The process repeats with each bit of **BIN** being introduced to **BCDACCUM** and then being scaled up on successive passes through the loop. Figure 7 shows the implementation of this algorithm for the 8049.

LOC	OBJ	SEQ	SOURCE STATEMENT	LOC	OBJ	SEQ	SOURCE STATEMENT
0001	A9	= 53	MOV R1, A	0011	28	= 70	XCH A, R0
0002	28	= 54	XCH A, R0	0012	A9	= 71	MOV R1, A
0003	BC03	= 55	MOV ICNT, #DIGPR	0013	28	= 72	XCH A, R0
0005	B100	= 56	BCDCOR: MOV @R1, #00	0014	BC03	= 73	MOV ICNT, #DIGPR
0007	19	= 57	INC R1	0016	A0	= 74	MOV TEMP1, A
0008	EC05	= 58	DJNZ ICNT, BCDCOR	0017	F1	= 75	BCDOC: MOV A, @R1
0009	8B10	= 59	:1 COUNT:=16	0018	71	= 76	ADDC A, @R1
000A	8B10	= 60	MOV COUNT, #16	0019	57	= 77	DA A
000B	97	= 61	:1 REPEAT	001A	A1	= 78	MOV @R1, A
000C	97	= 64	CLR C	001B	19	= 79	INC R1
000D	F7	= 65	RLC A	001C	EC17	= 80	DJNZ ICNT, BCDOC
000E	2A	= 66	XCH A, XA	001E	FD	= 81	MOV A, TEMP1
000F	F7	= 67	RLC A	001F	F624	= 83	JC BCDCOR
0010	2A	= 68	XCH A, XA	0021	EB0C	= 86	DJNZ COUNT, BCDCOR
		= 69	:2 BCD:=BCD*2+CARRY	0023	97	= 87	CLR C ; CLEAR CARRY TO INDICATE NORMAL TERMINATION
		= 70	XCH A, R0	0024	83	= 89	BCDCOR: RET
		= 71	MOV R1, A			= 90	END
		= 72	XCH A, R0				
		= 73	MOV ICNT, #DIGPR				
		= 74	MOV TEMP1, A				
		= 75	BCDOC: MOV A, @R1				
		= 76	ADDC A, @R1				
		= 77	DA A				
		= 78	MOV @R1, A				
		= 79	INC R1				
		= 80	DJNZ ICNT, BCDOC				
		= 81	MOV A, TEMP1				
		= 82	:2 IF CARRY FROM BCDOC GOTO ERROR EXIT				
		= 83	JC BCDCOR				
		= 84	:2 COUNT:=COUNT-1				
		= 85	:1 UNTIL COUNT=0				
		= 86	DJNZ COUNT, BCDCOR				
		= 87	CLR C ; CLEAR CARRY TO INDICATE NORMAL TERMINATION				
		= 88	:1 END CONVERT_TO_BCD				
		= 89	BCDCOR: RET				
		= 90	END				

USER SYMBOLS

BCDCOR 0005 BCDCOR 000C BCDCOR 0024 BCDCOR 0017 CNBCD 0000 COUNT 0003 DIGPR 0003 ICNT 0004
TEMP1 0005 XA 0002

ASSEMBLY COMPLETE, NO ERRORS

Figure 7 (continued)

```

COUNT:= DIGNO
REPEAT
  BCDACCUM:= BCDACCUM * 10
  BIN:= 10 * BIN + CARRY DIGIT
  COUNT:= COUNT - 1
UNTIL COUNT=0
END CONVERT_TO_BINARY

```

The only complexity is the two multiplications by ten. The BCDACCUM can be multiplied by ten by shifting it left four places (one digit). The variable BIN could be multiplied using the multiply algorithm already discussed, but it is usually more efficient to do this by mak-

This implies that the value $10 * \text{BIN}$ can be generated by saving the value of BIN and then shifting BIN two places left. After this the original value of BIN can be added to the new value of BIN (forming $5 * \text{BIN}$) and then BIN can be multiplied by two. It is often possible to implement the multiplication of a value by a constant by using such techniques. Figure 8 shows an 8049 routine which converts BCD values to binary. This routine differs slightly from the algorithm above in that the BCD digits are read, and converted to binary, two digits at a time. Protection has also been added to detect BCD operands which, if converted, would yield binary values beyond the range of the result.

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

LOC	OBJ	SEQ	SOURCE STATEMENT
		1	\$MACROFILE
		2	\$INCLUDE(:F1:CONBIN.HED)
		= 3	*****
		= 4	;
		= 5	;
		= 6	CONBIN
		= 7	*****
		= 8	;
		= 9	THIS UTILITY CONVERTS A 6 DIGIT BCD VALUE TO BINARY
		= 10	AT ENTRY:
		= 11	R0= POINTER TO A PACKED BCD STRING
		= 12	AT EXIT:
		= 13	A = LOWER EIGHT BITS OF THE BINARY RESULT
		= 14	XA= UPPER EIGHT BITS OF THE BINARY RESULT
		= 15	C = SET IF OVERFLOW ELSE CLEARED
		= 16	*****
		= 17	*****
		= 18	*****
		= 19	*****
		= 20	*****
		= 21	\$INCLUDE(:F1:CONBIN.PDL)
		= 22	;
		= 23	;
		= 24	1 CONVERT_TO_BINARY
		= 25	1 POINTER0:=POINTER0+DIGITPAIR-1
		= 26	1 COUNT:=DIGITPAIR
		= 27	1 BIN:=0
		= 28	1 REPEAT
		= 29	2 BIN:=BIN*10
		= 30	2 BIN:=BIN+MEM(R0)[7-4]
		= 31	2 BIN:=BIN*10
		= 32	2 BIN:=BIN+MEM(R0)[3-0]

LOC	OBJ	SEQ	SOURCE STATEMENT
		= 33 ; 2	POINTER0:=POINTER0-1
		= 34 ; 2	COUNT:=COUNT-1
		= 35 ; 1	UNTIL COUNT=0
		= 36 ; 1	END CONVERT_TO_BINARY
		37 ;	
		38 ;	EQUATES
		39 ;	=====
		40 ;	
0002		41 XA	EQU R2
0003		42 COUNT	EQU R3
0004		43 ICNT	EQU R4
		44 ;	
0003		45 DIGPR	EQU 3
		46 ;	
		47 \$EJECT	
		48 \$INCLUDE(:F1:CONBIN)	
		= 49 ;	
0005		= 50 TEMP1	SET R5
0006		= 51 TEMP2	SET R6
		= 52 ;	
		= 53 ; 1	CONVERT_TO_BINARY
		= 54	CONBIN:
		= 55 ; 1	POINTER0:=POINTER0+DIGITPAIR-1
0000 F8		= 56	MOV A,R0
0001 0302		= 57	ADD A,#DIGPR-1
0003 A8		= 58	MOV R0,A
		= 59 ; 1	COUNT:=DIGITPAIR
0004 BB03		= 60	MOV COUNT,#DIGPR
		= 61 ; 1	BIN:=0
0006 27		= 62	CLR A
0007 AA		= 63	MOV XA,A
		= 64 ; 1	REPEAT
		= 65	CONBLP:
		= 66 ; 2	BIN:=BIN*10
0008 142B		= 67	CALL CONB10
000A F62A		= 68	JC CONBER
		= 69 ; 2	BIN:=BIN+MEM(R0)[7-4]
000C AD		= 70	MOV TEMP1,A
000D F0		= 71	MOV A,@R0
000E 47		= 72	SWAP A
000F 530F		= 73	ANL A,#0FH
0011 6D		= 74	ADD A,TEMP1
0012 2A		= 75	XCH A,XA
0013 1300		= 76	ADDC A,#00
0015 2A		= 77	XCH A,XA
0016 F62A		= 78	JC CONBER
		= 79 ; 2	BIN:=BIN*10
0018 142B		= 80	CALL CONB10
001A F62A		= 81	JC CONBER
		= 82 ; 2	BIN:=BIN+MEM(R0)[3-0]
001C AD		= 83	MOV TEMP1,A
001D F0		= 84	MOV A,@R0
001E 530F		= 85	ANL A,#0FH
0020 6D		= 86	ADD A,TEMP1
0021 2A		= 87	XCH A,XA

LOC	OBJ	SEQ	SOURCE STATEMENT	LOC	OBJ
0022	1300	= 88	ADDC A, #00		
0024	2A	= 89	XCH A, XA		
0025	F62A	= 90	JC CONBER		
		= 91 ; 2	POINTER0:=POINTER0-1		
0027	C8	= 92	DEC R0		
		= 93 ; 2	COUNT:=COUNT-1		
		= 94 ; 1	UNTIL COUNT=0		
0028	EB08	= 95	DJNZ COUNT, CONBLP		
		= 96 ; 1	END CONVERT_TO_BINARY		
002A	83	= 97	CONBER: RET		
		= 98	\$EJECT		
		= 99 ;			
		= 100 ;			
		= 101 ;	UTILITY TO MULTIPLY BIN BY 10		
		= 102 ;	CARRY WILL BE SET IF OVERFLOW OCCURS		
		= 103 ;			
002B	AD	= 104	CONB10: MOV TEMP1, A ; SAVE A		
002C	2A	= 105	XCH A, XA ; SAVE XA		
002D	AE	= 106	MOV TEMP2, A		
002E	2A	= 107	XCH A, XA		
		= 108 ;			
002F	97	= 109	CLR C		
0030	F7	= 110	RLC A ; BIN:=BIN*2		
0031	2A	= 111	XCH A, XA		
0032	F7	= 112	RLC A		
0033	2A	= 113	XCH A, XA		
0034	F646	= 114	JC CONB1E ; ERROR ON OVERFLOW		
		= 115 ;			
0036	F7	= 116	RLC A ; BIN:=BIN*4		
0037	2A	= 117	XCH A, XA		
0038	F7	= 118	RLC A		
0039	2A	= 119	XCH A, XA		
003A	F646	= 120	JC CONB1E ; ERROR ON OVERFLOW		
		= 121 ;			
003C	6D	= 122	ADD A, TEMP1 ; BIN:=BIN*5		
003D	2A	= 123	XCH A, XA		
003E	7E	= 124	ADDC A, TEMP2		
003F	2A	= 125	XCH A, XA		
0040	F646	= 126	JC CONB1E ; ERROR ON OVERFLOW		
		= 127 ;			
0042	F7	= 128	RLC A ; BIN:=BIN*10		
0043	2A	= 129	XCH A, XA		
0044	F7	= 130	RLC A		
0045	2A	= 131	XCH A, XA		
		= 132 ;			
0046	83	= 133	CONB1E: RET		
		= 134			
		= 135 ;			
		136	END		
USER SYMBOLS					
CONB10	002B	CONB1E	0046	CONBER	002A
TEMP1	0005	TEMP2	0006	XA	0002
ASSEMBLY COMPLETE, NO ERRORS					
CONBLP	0008	COUNT	0003	DIGPR	0003
				1CNT	0004

CONCLUSION

The design goals of the full duplex serial communications software were realized; if transmission and reception are occurring concurrently, only 42 percent of the real time available to the 8049 will be consumed by the serial link. This implies that an 8049 running full duplex serial I/O will still outperform earlier members of the family running without the serial I/O requirement. It is also possible to run this program in an 8048 or 8748 at 1200 baud with the same 42 percent CPU utilization.

The execution times for the other routines that have been discussed have been summarized in Table 1. All of these routines were written to maintain maximum usability rather than minimum code size or execution time. The resulting execution times and code size are therefore what the user can expect to see in a real application. The results that were obtained clearly show the efficiency and speed of the 8049. The equivalent times for the 8048 are also shown. It is clear that the 8049 represents a substantial performance advantage over the 8048. Considering, in most applications, that the 8048 is

the highest performance microcomputer available to date, the performance advantage of the 8049 should allow the cost benefits of a single chip microcomputer to be realized in many applications which up until now have required too much "computer power" for a single chip approach.

	EXECUTION TIME (MICROSECONDS)		
	BYTES	8049	8048
MPY8	21	109	200
DIV 16	37	183 MIN 204 MAX	335 MIN 375 MAX
CONBCD	36	733	1348
CONBIN	70	388	713

Table 1. Program Performance

A High-Speed Emulator for Intel MCS-48[®] Microcomputers

Contents

I. PURPOSE AND SCOPE	3-84
II. THE HSE-49™ DEVELOPMENT TOOL	3-84
III. GENERAL HARDWARE OVERVIEW	3-84
IV. INTERPROCESSOR COMMUNICATION	3-86
V. HSE-49 COMMAND DESCRIPTION	3-87
VI. SYSTEM LIMITATIONS	3-90
VII. HARDWARE CONFIGURATIONS	3-91
APPENDIX A. SCHEMATIC DIAGRAMS	3-93
APPENDIX B. MONITOR LISTINGS	3-97
APPENDIX C. COMMAND SUMMARY	3-184
APPENDIX D. ERROR MESSAGES	3-184

The primary intent of note is to provide the reader with the information needed to reconstruct and make full use of the HSE-49 emulator. Less emphasis is placed on describing how the hardware operates or how the commands are implemented. This information may be found in the schematic diagrams and software listings included in the Appendices.

The main concern in designing the HSE-49 emulator was to keep the basic design simple, while maximizing the system's flexibility. The design allows the use of jumpers, hardware and software switches, etc. to allow the user to reconfigure the system according to the way he dedicates chip select pins, I/O, etc. The emulator can be changed to fit each user's unique needs, rather than forcing the user to alter his needs to what is provided.

While the HSE-49 emulator can assist a new microcomputer user in becoming familiar with the 8048 and 8049 microcomputers, its inherent debug capabilities will also prove helpful to design engineers. The design could be used for new system development and verification or adapted for prototype production.

III. GENERAL HARDWARE OVERVIEW

User Program Emulation

The actual emulation of the user's program is done using an 8038 microcomputer (IC29) on the schematic in Appendix A) executing a program stored in external RAM. The basic minimum configuration includes the 8038 microcomputer, an 8285 address latch (IC18), and 2K bytes of 2114 RAM to use for program development and real-time execution (ICs B1, C1, B2, and C2). Additional RAM may be added to allow the user to expand his program and data memory to 4K each. (In 11-MHz crystal is used with the microcomputer, type 2114-3 RAMs must be used.)

II. THE HSE-49 DEVELOPMENT TOOL

In essence, the HSE-49 emulator provides the user a means for executing an MCS-48 program located in external RAM rather than internal ROM or EPROM. This allows programs being debugged to be modified easily and quickly during the debug cycle. A user's program may be entered into system RAM either manually or via a serial link from a host computer such as an Intel® Microcomputer Development System. Once loaded, the program can be modified using an on-board keyboard and display, and executed in real-time in a number of breakpoint modes. The internal state of the processor, including RAM, accumulator, timer/counter, and status register contents, can also be read and modified through the keyboard.

Breakpoint and debug facilities are extremely flexible. The following execution modes are provided:

- Programs may be run in full (11 MHz) real time;
- Programs may be single-stepped;
- In break mode, programs run in full real time until break occurs;

I. PURPOSE AND SCOPE

This Application Note presents a description of the design and operation of a high-speed emulator for the Intel® MCS-48™ family of single chip microcomputers. The HSE-49™ emulator provides a simple and inexpensive means for executing and debugging 8049 programs which require the full 11-MHz operating speed of the part.

Section II of this Application Note describes some of the features of this development tool and how it may be used. Section III briefly discusses the hardware used to implement these features, while Section IV describes the manner in which program execution status is made available to the operator.

A detailed description of all of the operator commands is presented in Section V of this note, along with the modifiers and options which may be specified for each command. Known restrictions and limitations of the HSE-49 system are listed and explained in Section VI. Section VII shows how the basic circuit may be modified to provide options on memory organization, I/O configurations, etc.

Full schematics of the system hardware, as well as monitor software listings, are presented in Appendices A and B, respectively. A short summary of the command syntax is presented in Appendix C. Appendix D explains the error message codes which may appear during use.

It is assumed that the reader is already familiar with the operation of the 8048 or 8049 microcomputers. Some knowledge of the 8048 architecture is needed to understand sections of the command and modifier descriptions. Most users will already have this background. Other readers are referred to the *MCS-48 Microcomputer User's Manual*, Intel publication number 9800270.

II. THE HSE-49 DEVELOPMENT TOOL

In essence, the HSE-49 emulator provides the user a means for executing an MCS-48 program located in external RAM rather than internal ROM or EPROM. This allows programs being debugged to be modified easily and quickly during the debug cycle. A user's program may be entered into system RAM either manually or via a serial link from a host computer such as an Inteltec® Microcomputer Development System. Once loaded, the program can be modified using an on-board keyboard and display, and executed in real-time in a number of breakpoint modes. The internal state of the processor, including RAM, accumulator, timer/counter, and status register contents, can also be read and modified through the keyboard.

Breakpoint and debug facilities are extremely flexible. The following execution modes are provided.

- Programs may be run in full (11 MHz) real time;
- Programs may be single-stepped;
- In break mode, programs run in full real time until break occurs;

- Breaks may be triggered by either program or external data RAM accesses;
- Any number of breakpoints may be used in any combination;
- "Auto-Step" operation causes the current program counter and Accumulator contents to be printed on the display for a short time on every instruction cycle;
- "Auto-Break" provides the above display only when a break flag is encountered, with real time operation otherwise;
- While running in non-break mode, a TTL-level pulse is generated whenever a break flag is encountered. This signal may be used to trigger an oscilloscope or Logic Analyzer to assist in hardware and software debug.
- While running in any mode, the keyboard and display are "alive". Execution may be suspended or terminated by commands from the keyboard.

Intent of this Note

While the HSE-49 emulator can assist a new microcomputer user in becoming familiar with the 8048 and 8049 microcomputers, its inherent debug capabilities will also prove helpful to design engineers. The design could be used for new system development and verification or adapted for prototype production.

The main concern in designing the HSE-49 emulator was to keep the basic design simple, while maximizing the system's flexibility. The design allows the use of jumpers, hardware and software switches, etc. to allow the user to reconfigure the system according to the way he dedicates chip-select pins, I/O, etc. The emulator can be changed to fit each user's unique needs, rather than forcing the user to alter his needs to what is provided.

The primary intent of note is to provide the reader with the information needed to reconstruct and make full use of the HSE-49 emulator. Less emphasis is placed on describing how the hardware operates or how the commands are implemented. This information may be found in the schematic diagrams and software listings included in the Appendices.

III. GENERAL HARDWARE OVERVIEW

User Program Emulation

The actual emulation of the user's program is done using an 8039 microcomputer (IC29 on the schematics in Appendix A) executing a program stored in external RAM. The basic minimum configuration includes the 8039 microcomputer, an 8282 address latch (IC19), and 2K bytes of 2114 RAM to use for program development and real-time execution (ICs B1, C1, B2, and C2). Additional RAM may be added to allow the user to expand his program and data memory to 4K each. (If an 11-MHz crystal is used with the microcomputer, type 2114-3 RAMs must be used.)

System Supervision

A second microcomputer — another 8039 (IC25) with an 8282 address latch (IC16) and off-chip program memory in a 2716 EPROM (IC15) — is used to scan the on-board keyboard and display, interpret and implement commands, drive serial interfaces, etc. In general, the master processor is used to interface the execution processor's memory spaces with the outside world and control the operation of the execution processor. In this note the two processors will be abbreviated "MP" and "EP", respectively. Figure 1 shows how the two processors interrelate with the rest of the system.

Keyboard/Display

The 33-key keyboard shown in Figure 2 includes a 16-key hexadecimal keypad and 17 special function keys for specifying commands and modifiers. Readers already

familiar with the PROMPT-48™ debug tool for the 8048 will find that 25 of the HSE-49 emulator keys are identical in function and layout to the PROMPT-48 keyboard, and use the PROMPT-48 command syntax. The eight additional keys are used to generalize and augment the PROMPT-48 capabilities, as described in Section V.

The eight-character seven-segment display (DS1-DS8) is used for displaying addresses, data, and pseudo-alphanumeric messages. The display responses printed in Section V and throughout this note use a mix of upper and lower case letters to indicate what seven-segment patterns appear. An 8243 (IC9) and eight DIP packages (resistor packs, current buffers, etc.) are used for multiplexing the display and scanning the keyboard.

Breakpoint Detection

Breakpoints are specified and detected using a 2102A 1K x 8 RAM corresponding to each pair of 2114s (ICs A1

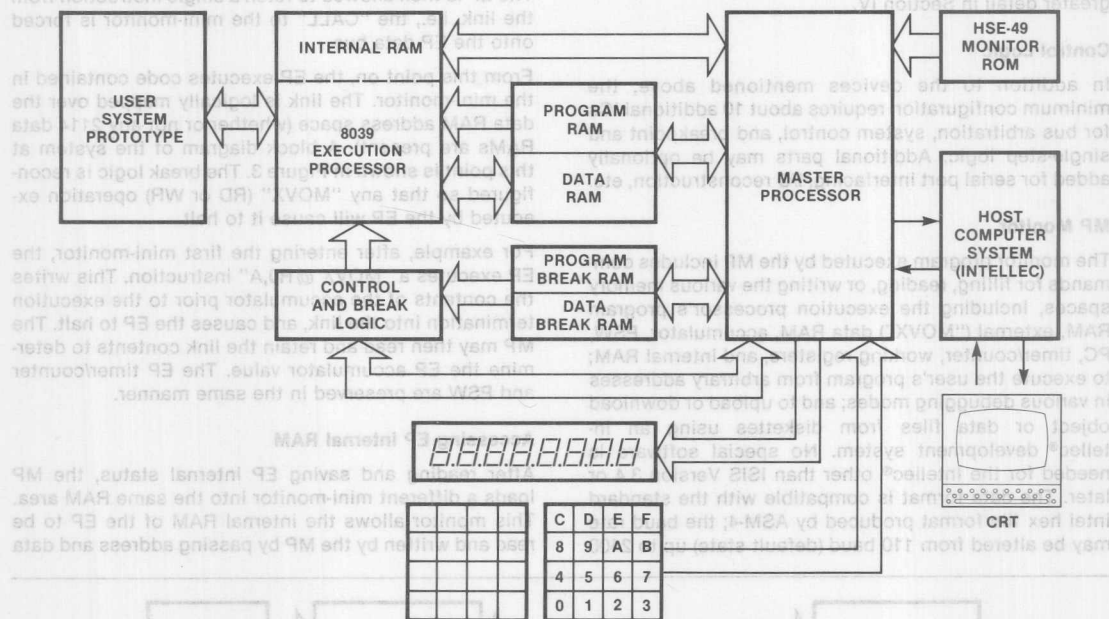


Figure 1. HSE-49™ Emulator Signal Flow Diagram



Figure 2. HSE-49™ Emulator Command Keyboard Organization

asserted, is latched (IC27 and IC36) and used to halt the execution processor via the single-step input. (In other modes, the break logic can be reconfigured to set the break requested flip-flop on any EP machine cycle or any EP "MOVX" instruction.)

Link Register

An 8212 8-bit latch (IC18) is used to communicate data and commands between the master and control processors. Under control of the MP, this register, called the "Link" register, may be logically mapped into either the program or data RAM address spaces. When this is done, the 2114s in the respective memory space are disabled and the link responds to all accesses, regardless of address. The link will be discussed in greater detail in Section IV.

Control Logic

In addition to the devices mentioned above, the minimum configuration requires about 10 additional ICs for bus arbitration, system control, and breakpoint and single-step logic. Additional parts may be optionally added for serial port interfacing, I/O reconstruction, etc.

MP Monitor

The monitor program executed by the MP includes commands for filling, reading, or writing the various memory spaces, including the execution processor's program RAM, external ("MOVX") data RAM, accumulator, PSW, PC, timer/counter, working registers, and internal RAM; to execute the user's program from arbitrary addresses in various debugging modes; and to upload or download object or data files from diskettes using an Inteltec® development system. No special software is needed for the Inteltec® other than ISIS Version 3.4 or later. The data format is compatible with the standard Intel hex file format produced by ASM-4; the baud rate may be altered from 110 baud (default state) up to 2400

IV. INTERPROCESSOR COMMUNICATION

Program Break Sequence

When the MP detects that the EP has been halted by the breakpoint hardware, or when the operator presses a key while the program is executing, the program break sequence is initiated. The low-order 23 bytes of user program memory is read into a buffer within the internal RAM of the MP. A short program for reading and transmitting internal EP status is written over the low-order program memory. (This is one of several "mini-monitors" overlayed over the user program area.) The link register is mapped logically over the user program memory, and loaded with the 8049 machine code for a "CALL" instruction to the mini-monitor program area. The EP is then allowed to fetch a single instruction from the link, i.e., the "CALL" to the mini-monitor is forced onto the EP data bus.

From this point on, the EP executes code contained in the mini-monitor. The link is logically mapped over the data RAM address space (whether or not any 2114 data RAMs are present). A block diagram of the system at this point is shown in Figure 3. The break logic is reconfigured so that any "MOVX" (RD or WR) operation executed by the EP will cause it to halt.

For example, after entering the first mini-monitor, the EP executes a "MOVX @R0,A" instruction. This writes the contents of the accumulator prior to the execution termination into the link, and causes the EP to halt. The MP may then read and retain the link contents to determine the EP accumulator value. The EP timer/counter and PSW are preserved in the same manner.

Accessing EP Internal RAM

After reading and saving EP internal status, the MP loads a different mini-monitor into the same RAM area. This monitor allows the internal RAM of the EP to be read and written by the MP by passing address and data

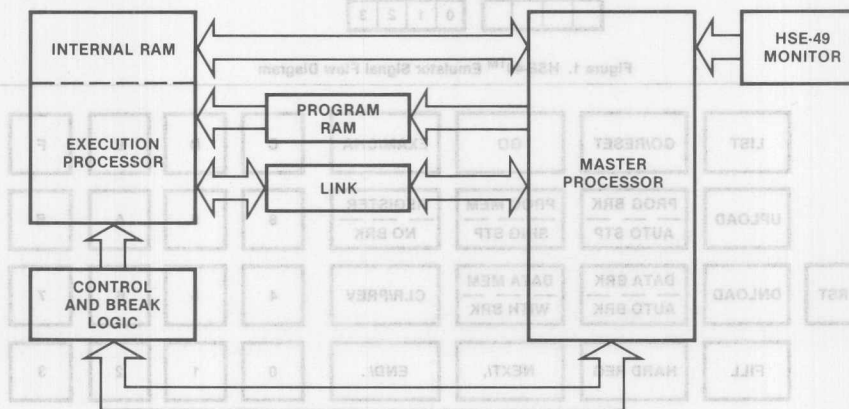


Figure 3. Communication between EP & MP

values between the two processors using the link register.

This is needed for two reasons. First, the EP program counter prior to the forced "CALL" instruction may be derived from the EP stack contents, and may be modified to cause the EP to resume execution at any desired address. Secondly, the internal RAM of the EP may then be accessed and modified in the process of executing a number of the monitor commands.

Resuming User Program Execution

In order to resume user program execution, a status-restoration mini-monitor is overlaid. This restores the EP internal status using a scheme analogous to the one in which the status was originally saved. The final step of the last mini-monitor is an "RETR" instruction, after which the EP is again halted. The low-order program memory saved earlier is rewritten into the appropriate area, the break logic is reconfigured for the desired execution mode, and the EP is released to run at full speed until the next break situation is encountered.

Note that all commands are implemented using "logical" rather than "physical" addressing. Thus the operator need not be concerned with the intricacies of the system design. For example, when any monitor command refers to low-order user program memory, the appropriate byte of storage within the MP internal RAM is accessed instead. If the location is altered, the internal RAM is modified appropriately. When program memory is reloaded prior to resuming user program execution, the modified version of the user program will be the one loaded.

Baud	HR06	HR07
110	93H	04H
150	96H	03H
300	45H	02H
600	9DH	01H
1200	44H	01H
2400	1AH	01H

Table 1. Serial Interface Data Rate Parameters

V. HSE-49 COMMAND DESCRIPTION

Whenever the characters "HSE-49" are present on the system display, a command string may be entered by the operator. In general, all command strings consist of a basic command initiator, an optional command modifier or type-designator, and a number of parameters or delimiters entered as hexadecimal digits. A command is executed, or a command in progress terminated, by pressing the [END/] key. Logical default values are assumed for the modifier and parameters if either (or both) are omitted. A default parameter assumed for the command modifier will be presented on the display when the first parameter is entered.

Each parameter is a string of up to three hexadecimal digits. If more than three digits are entered, only the most recent three are considered. This allows an erroneous digit to be corrected without respecifying the entire command. A parameter is completed by pressing the [NEXT/] key. Some commands may only need the

low order part of a parameter; i.e., a command incorporating a data byte (such as [FILL]) will use only the low-order 8 bits of the corresponding parameter; Internal RAM and hardware register addressing uses only seven. In each case, higher order bits are ignored.

A command string is terminated and the command invoked by pressing the [END/] key. The command will also be invoked by pressing the [NEXT/] key when no additional parameters are allowed. A command string may be aborted at any point before the command is invoked by pressing the [CLEAR/PREV] key, and the sign-on message will appear.

Errors

An illegal command string, command terminator, or hardware failure will cause an error message and error code number to appear on the display (e.g., "Error:3"). When this occurs, the monitor can be returned to command mode by pressing the [CLEAR] or [END/] keys. An explanation of the various error codes is given in Appendix D.

Command Classes

Commands for the HSE-49 emulator are divided into general classes, where all commands in each class have the same choice of options or modifiers. A brief description of each command, followed by a description of the allowed options, is presented below by class.

Data Manipulation/Control Command Group

Commands:

[EXAM/CHA]

Display Response — "ECh."

Function — Examine/change memory location.

Causes the memory address specified to be read and presented on the display. New data may be entered (if desired) from the hexadecimal keypad. New data is verified before appearing on the display. Subsequent or previous locations may be read by pressing the [NEXT/] or [PREV] keys, respectively. Command terminated with [END/] key.

[FILL]

Display Response — "FIL."

Function — Fill range of memory addresses with a single data value.

Fill the appropriate memory space between the addresses specified by the first two parameters with the low-order byte of the third parameter. If second parameter less than first, only the location specified by the first is affected. If third parameter omitted, zero is assumed. If second and third parameters omitted, individual address specified is cleared. Command is useful for setting a large range of breakpoints; e.g., all of page 3 may be enabled for break with the command:

[FILL][PROG BRK]<300>[,<3FF>[,<1>[.]

[LIST]

Display Response — "LSI."

Function — List memory to output device through HSE-49 serial port.

Display the contents of a range of addresses given by two parameters to a teletype or CRT screen. Data is formatted, 16 separated bytes per line, with the starting address of each line printed. If used with an Intellec® system, the operator first uses ISIS-II to transfer the TTY input to the CRT output ("COPY :TI: TO :CO:") then invokes this command from the keypad. Alternatively, any ISIS device or disk file name(:TO:, :LP:, :F1:HRDREG.SAV, etc.) may be used as the destination.

[DNLOAD]

Display Response — "dnL."

Function — Download memory through HSE-49 serial port

Load data in hex file format through the serial input port. If used with Intellec® system, the operator first invokes this command from the keypad, then uses ISIS-II to transfer a disk file to the teletype port ("COPY :Fn:file.HEX TO :TO:").

The use of the checksum field for the download command is expanded slightly over the Intel hex file format standard. If the first character of the checksum field is a question mark ("?"), the checksum for that record will not be verified. This allows large object files produced by the assembler to be patched using the ISIS text editor without the necessity of manually recomputing the checksum value.

[UPLOAD]

Display Response — "UPL."

Function — Upload memory through HSE-49 serial port.

Output the contents of a range of addresses specified by the two parameters through the HSE-49 serial port in standard Intel hex file format. If used with Intellec® system, the operator first uses ISIS-II to transfer the TTY input to a disk file ("COPY :TI: TO :Fn:file.HEX"), then invokes this command from the keypad.

Data types allowed:

[PROG MEM]

Display Response — "Pr."

Function — User program memory.

Memory used to develop and execute user program. Addresses 000 through 7FF are the execution processor's memory bank 0; 800 through FFF are memory bank 1.

[REGISTER]

Display Response — "rG."

Function — Register memory and RAM.

Internal RAM of execution processor. Locations 0-7 are working register bank 0; 18-1F are working register bank 1. Only the low-order 7 bits of an address are considered.

[DATA MEM]

Display Response — "dA."

Function — External data memory (if installed).

Memory accessed by execution processor "MOVX A,@Rr" or "MOVX @Rr,A" instructions. High-order 4 bits may or may not be relevant, depending on jumpering option selected (explained in Section VII of this note).

[HARD REG]

Display Response — "Hr."

Function — Hardware registers.

The execution processor (EP) hardware registers (accumulator, timer/counter, etc.), as well as several parameters for controlling HSE-49 system status, are accessible through this catch-all memory space. Addresses are as follows:

00 — EP accumulator.

01 — EP PSW.

Bits correspond to 8049 PSW except that bit 3 (unused in the 8049) is used to monitor and alter the state of F1. Bits 2-0 correspond to the stack pointer value after the EP executes a CALL to the mini-monitor; i.e., one greater than when EP was running the user's program.

02 — EP timer/counter.

03 — EP internal RAM location 00.

(This value is also accessible through [REGISTER] space.)

04 — EP program counter (low byte).

05 — EP program counter (high nibble).

06-07 — HSE-49 serial interface baud rate parameters. Defaults to 110 baud; other rates may be selected by loading the values listed in Table 1.

08 — HSE-49 automatic sequencing rate parameter. Used in [GO][AUTO STP] and [GO][AUTO BRK] execution commands. 00 → fastest; FF → slowest. Defaults to 20H; approximately two steps per second.

09 — Monitor version/release number (packed BCD).

0A-0F — Currently unused by the monitor program.

10-7F — Variables used by master processor (MP) monitor. Should not be altered by operator.

[PROG BRK]

Display Response — "Pb."

Function — User program breakpoint memory.

Memory space used to indicate points where program execution should halt when running in a mode with breakpoints enabled ([GO][W/ BRK] and [GO][AUTOBRK]). Break will occur if enabled byte is read as the first or last byte of a 2-byte instruction, or read in executing a MOVP, MOV3, or JMPP instruction. Memory is only 1 bit per location; 00 indicates continue, 01 causes a halt. Addresses 000 through 7FF are the execution processor's memory bank 0; 800 through FFF are memory bank 1.

[DATA BRK]

Display Response — "db."

Function — External data RAM breakpoint memory.

Memory space used to indicate points where data accesses should halt when running in a mode with breakpoints enabled ([GO][W/ BRK] and [GO][AUTOBRK]). Memory is only 1 bit per location; 00 indicates continue, 01 causes a halt. High-order 4 bits of breakpoint address may or may not be relevant, dependent on jumpering option selected for the corresponding data RAM (explained in Section VII of this note).

User Program Execution Control Group

Commands:

[GO]

Display Response — "Go."

Function — Begin execution.

If a parameter is given as part of the command string, execution will begin at that address. Otherwise, the EP program counter (hardware registers 04 and 05) will be used. These will contain the program counter from an earlier program execution break unless they have since been explicitly modified by the operator.

If command is terminated by [END/], the EP's F1, PSW and stack pointer will be cleared. If command string is terminated by [NEXT/], PSW will be taken from the EP PSW contents (hardware register 01).

While running the user's program, the characters "-run-" are written on the display. Execution may be halted and another command initiated by pressing the appropriate command key. Execution may be suspended at any time in any mode by pressing the [END/] key. This will cause the current value of the execution processor program counter and accumulator to appear on the display in the form "PC.234-56". System status is saved in the appropriate hardware registers. At this point, or when an enabled breakpoint is encountered, pressing the [NEXT/] key will cause the program to continue in the same mode as before. Any other command may be invoked by pressing the appropriate command string.

[GO/RESET]

Display Response — "Gr."

All mnemonics copyrighted © Intel Corporation 1976.

Function — Go from reset state.

EP is hardware-reset and released to execute the user's program from location 000H. No parameters are allowed. F0, F1, PSW, stack pointer, memory bank flip-flop, etc., are cleared.

Note that this command does not require the use of mini-monitors to initiate program execution. As the last phase of the program development cycle, the 2114 program RAMs and address decoder may be removed and replaced by a ROM or EPROM part (not shown in schematics). This command may be used to start execution when the program RAM has been removed. No interrogation of EP status or internal RAM may be done, nor are break or single-step modes allowed in this case, though the 2102A breakpoint RAM outputs may still be used to trigger a logic analyzer.

Execution modes allowed:

[NO BRK]

Display Response — "nb."

Function — Without breakpoints.

Full-speed execution without breakpoints enabled. Does not affect the state of the breakpoint memories.

[SING STP]

Display Response — "SSt."

Function — Single Step.

Step through program one instruction at a time. After each instruction is executed, execution halts with the current value of the Execution Processor Program Counter and Accumulator appearing on the display in the form "PC.234-56". System status is saved in the appropriate Hardware Registers. At the point, [NEXT/] will cause the program to execute one more instruction, or any other command may be invoked by pressing the appropriate command string. Does not affect the state of the Breakpoint Memories.

[W/ BRK]

Display Response — "br."

Function — With breakpoints.

Full-speed execution with breakpoints enabled. When a breakpoint is encountered, execution halts with the current value of the execution processor program counter and accumulator appearing on the display in the form "PC.234-56". System status is saved in the appropriate hardware registers. At this point, [NEXT/] will cause the program to continue until the next breakpoint is reached, or any other command may be invoked by pressing the appropriate command string.

[AUTO STP]

Display Response — "ASt."

Function — Automatically sequence through a series of instructions.

display in the form "PC.234-56". System status is saved in the appropriate hardware registers. Execution resumes after a time determined by contents of hardware register 08. Does not affect the state of the breakpoint memories.

[AUTO BRK]

Display Response — "Abr."

Function — Automatically sequence between breakpoints.

Execute a series of instructions in real time between breakpoints. When breakpoint is encountered, halt EP temporarily while program counter and accumulator contents are displayed, then continue. Display is sustained after execution resumes. Does not affect the state of the breakpoint memories.

Breakpoint Control Command Group

Commands:

[B]

Display Response — "Stb."

Function — Breakpoint set.

Set breakpoint for the address given. Multiple breakpoints may be set by entering additional addresses, separated by the [NEXT/.] key. Command terminated by pressing [END/]. Action taken is to fill the appropriate breakpoint memory locations with logical ones.

[C]

Display Response — "CLb."

Function — Clear breakpoint.

Clear breakpoint for the address given. Multiple breakpoints may be cleared by entering additional addresses, separated by the [NEXT/.] key. Command terminated by pressing [END/]. Action taken is to fill the appropriate breakpoint memory locations with logical zeroes.

Data types allowed:

[PROG MEM]

Display Response — "Pr."

Function — Break on program memory fetch.

Applies command to the program breakpoint memory space.

[DATA MEM]

Display Response — "dA."

Function — Break on data memory access.

Applies command to the external data breakpoint memory space.

Display Response — "HSE-49."

Function — System reset.

Reset both the MP and EP and clear all breakpoints (requires approximately one second). CAUTION — If reset while EP is executing the user's program, the low order section of program memory (about 23 bytes) will be altered.

VI. SYSTEM LIMITATIONS

In designing the HSE-49 emulator, certain compromises were made in an attempt to maximize the usefulness of the emulator while keeping the circuitry simple and inexpensive. As a result, the following limitations exist and must be taken into account when using the system.

1. As explained in Section IV, user program execution is terminated (by single-stepping, breakpoints, pressing the [END/.] key, etc.) by forcing the execution processor to execute a "CALL" instruction to the mini-monitor. This uses one level of the EP subroutine stack. The EP PSW reflects the value of the stack pointer *after* processing this CALL. As a result, the value indicated for stack depth by examining the EP PSW (hardware register 0t) is one greater than the depth when the break was initiated. The user program must not be using all eight levels of stack when a break is initiated or the bottom level will be destroyed.
2. User program is initiated (by the [GO] command or when resuming execution after a breakpoint, single-stepping, etc.) by forcing the EP to execute an "RETR" instruction. This will clear the EP interrupt-in-progress flip-flop. If the user program allows both external and timer interrupts to be enabled at the same time, care must be taken to avoid causing a break while the EP is within an interrupt servicing routine. No limitation is placed on breakpoints or single-stepping in the background program because of this.
3. When the user program execution is terminated (by a break, single-stepping, etc.) and later resumed, the EP timer/counter is restored to its value when the break occurred (unless modified by the user). The prescaler, however, will have changed. Thus, up to 31 machine cycles may be "lost" or "gained" if a break occurs while the timer is running.
4. Timer interrupts occurring at the same time as an EP break may be ignored if the timer overflow occurs after breaking user program execution before the timer value is saved.
5. The 8049 "RET" and "RETR" instructions are each 1-byte, 2-cycle instructions. During the second cycle the byte following the return instruction is fetched and ignored. If a program breakpoint is set for a location following a "RET" or "RETR" instruction, a break will be initiated when the return is executed.

6. Breakpoints should not be placed in the last 3 bytes of an EP memory bank (locations 7FDH-7FFH and 0FFDH-0FFFH). User program should not be single-stepped or auto-stepped through these locations.
7. Since I/O configuration is determined by external hardware rather than software, I/O modes may not be altered while a program is executing. (See Section VII for further details.)
8. The "ANL BUS,#nn" and "ORL BUS,#nn" instructions may not be used in the user program, as external hardware cannot properly restore these functions.
9. The memory bank select flag is not affected by the user program break sequence. Upon resuming execution with the [GO] command this flag will remain in the same state as before the preceding break. The flag may be cleared only by executing the [GO/RESET] or [SYS RST] commands.

VII. HARDWARE CONFIGURATIONS

A number of control and status lines are available to the user. All are low-power Schottky TTL-compatible signals.

TP1 — Unused MP input.

TP2 — Unused MP output.

TP3 — User program suspended. Low when EP running user code. High when halted or running mini-monitors.

TP4 — Breakpoint encountered. Normally low. High-level pulse generated when breakpoint passed. Useful for triggering logic analyzers, oscilloscopes, etc.

TP5 & TP6 — Memory matrix mode control. Select program vs. data RAM, link mapping configuration, etc. (See Appendix B for details.)

TP7 — Bus control. Low when MP controls common memory buses. High when EP controls memory buses.

The HSE-49 emulator hardware is designed to allow the user to reconfigure the system for a wide variety of different applications by installing or removing jumper wires or additional components. The schematics in Appendix A show the components needed for a variety of different configurations. In general, not all of the devices are required (or allowed) for any one configuration. The devices which are required are included in the following description.

The types of options allowed are divided below into several general classes and subdivided into mutually-independent features. Within some of these features there are numbered, mutually exclusive configurations; i.e., the serial interface (if desired) may use either

current-loop or RS-232C current buffers, but not both at one time.

Standard Operating Configuration

(Minimum system configurations — up to 4K program RAM; no data RAM; no serial interfaces; no execution processor I/O reconstruction.)

A. Basic 2K monitor from Appendix B:

Install resistors R4-R6
Install transistor Q1
Install crystals Y1-Y2
Install capacitors C5-C38
Install switches S1-S33
Install displays DS1-DS8
Install IC1-IC2
Install RP3-RP5
Install IC6-IC7
Install RP8
Install IC9
Install IC15-IC20
Install IC25-IC30
Install IC34
Install IC36-IC38
Install A1-A2
Install B1-B2
Install C1-C3
Install jumpers 13-15
Install jumpers 17-18
Install jumper 20

B. Expansion 2K monitor:

Install IC14
Remove jumper 17

Serial Interface Buffer Selection

A. Current loop serial interfaces (4N46s) installed for use with full Intellec® Model 800 development system TTY port.

Install IC21-IC22
Install resistor R1-R3
Install jumpers 4-9
(Remove RS-232 jumpers)

B. RS-232C serial interfaces (MC1488 and MC1489) installed for use with CRT as output device for data dumps:

Install IC23-IC24
Install jumpers 1-3
Install jumpers 10-11
(Remove current-loop jumpers)

External Data RAM Address Decoding Scheme for Execution Processor

A. Up to 16 pages of on-board external data RAM installed for execution processor (addresses 0 through

0FFFH = 4K bytes); port 2 used for addressing pages 0 through 15:

Install jumpers 21-25.

Install jumper 27

Install A5-A8

Install B5-B8

Install C5-C8

B. One page of on-board external data RAM installed for execution processor (addresses 0 through 0FFFH); port 2 not used for data addressing:

Install jumper 26

Install jumper 28

Install A5

Install B5

Install C5

Connect the outputs of IC20, pins 7, 9, 10, & 11 to the inputs of a 74LS21 AND gate (not shown). Connect the output to CE and CS inputs of A5-C5. (Note: these signals are all present at jumpers 21-24 on the schematics.)

Reconstructing I/O for Execution Processor

A. Application of port 2, pins P23-P20:

(1) Using P23-P20 for latched output data (used with "OUTL P2,A", "ANL P2,#data", and "ORL P2,#data" instructions):

Install IC31

(2) Using P23-P20 for interfacing to an 8243 in user's prototype:

Connect D3-D0 pins on IC31 socket to corresponding Q3-Q0 pins.

B. Application of execution processor BUS:

(1) Use of BUS as latched output port ("OUTL BUS,A"):

Install IC32

A. Up to 16 pages of on-board external data RAM installed for execution processor (addresses 0 through 0FFFH = 4K bytes); port 2 used for addressing pages 0 through 15:

Install jumpers 21-25.

Install jumper 27

Install A5-A8

Install B5-B8

Install C5-C8

B. One page of on-board external data RAM installed for execution processor (addresses 0 through 0FFFH); port 2 not used for data addressing:

Install jumper 26

Install jumper 28

Install A5

Install B5

Install C5

Connect the outputs of IC20, pins 7, 9, 10, & 11 to the inputs of a 74LS21 AND gate (not shown). Connect the output to CE and CS inputs of A5-C5. (Note: these signals are all present at jumpers 21-24 on the schematics.)

Reconstructing I/O for Execution Processor

A. Application of port 2, pins P23-P20:

(1) Using P23-P20 for latched output data (used with "OUTL P2,A", "ANL P2,#data", and "ORL P2,#data" instructions):

Install IC31

(2) Using P23-P20 for interfacing to an 8243 in user's prototype:

Connect D3-D0 pins on IC31 socket to corresponding Q3-Q0 pins.

B. Application of execution processor BUS:

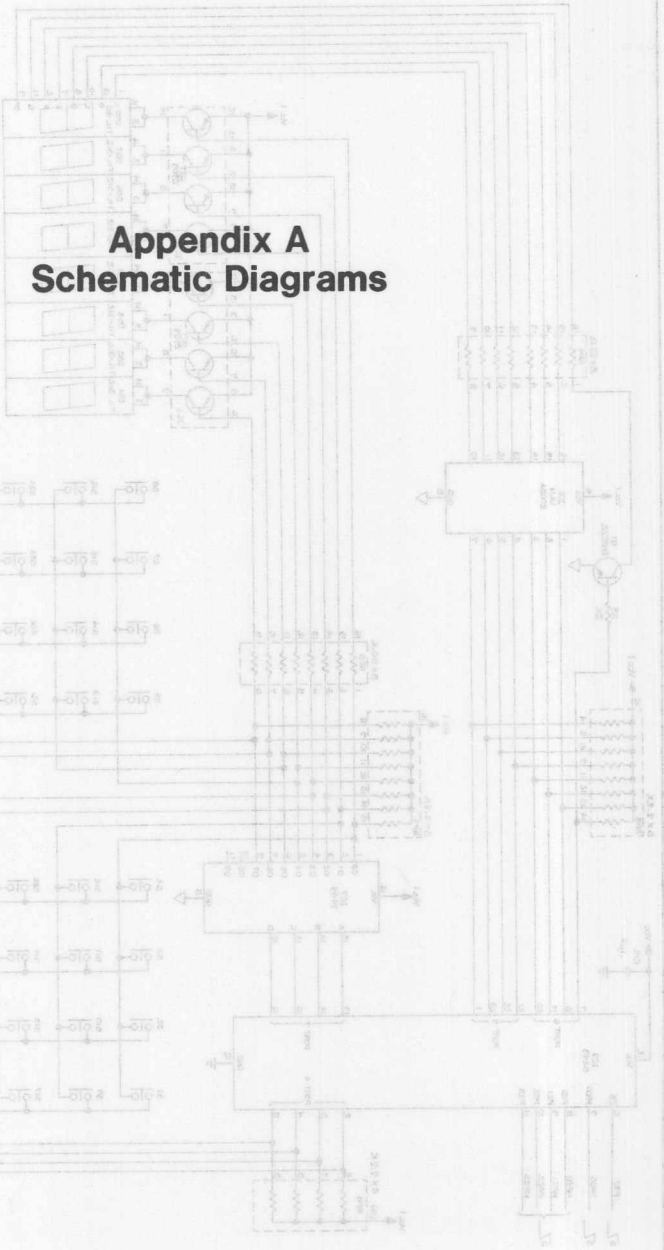
(1) Use of BUS as latched output port ("OUTL BUS,A"):

Install IC32

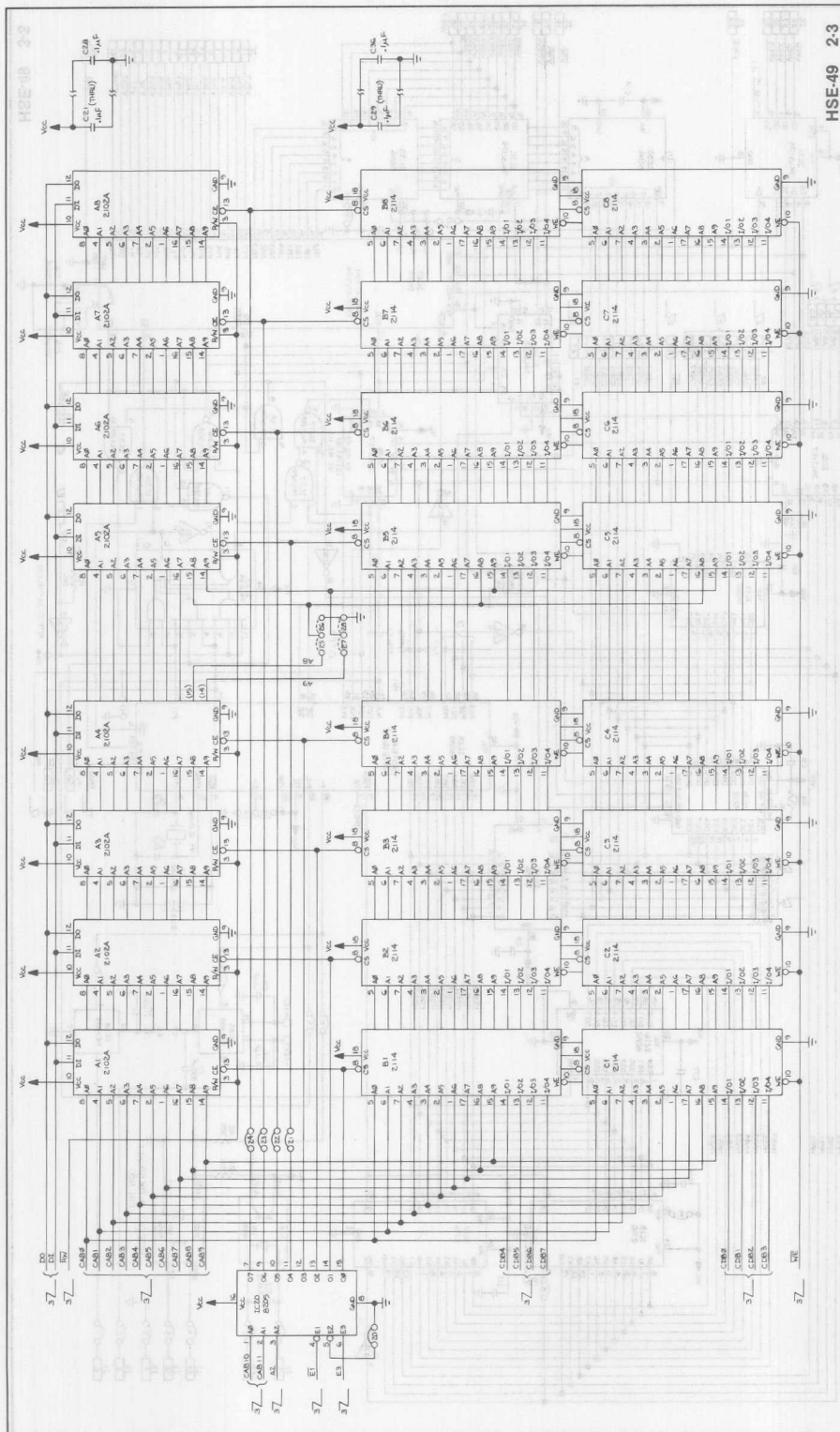
HSE-10 1/2



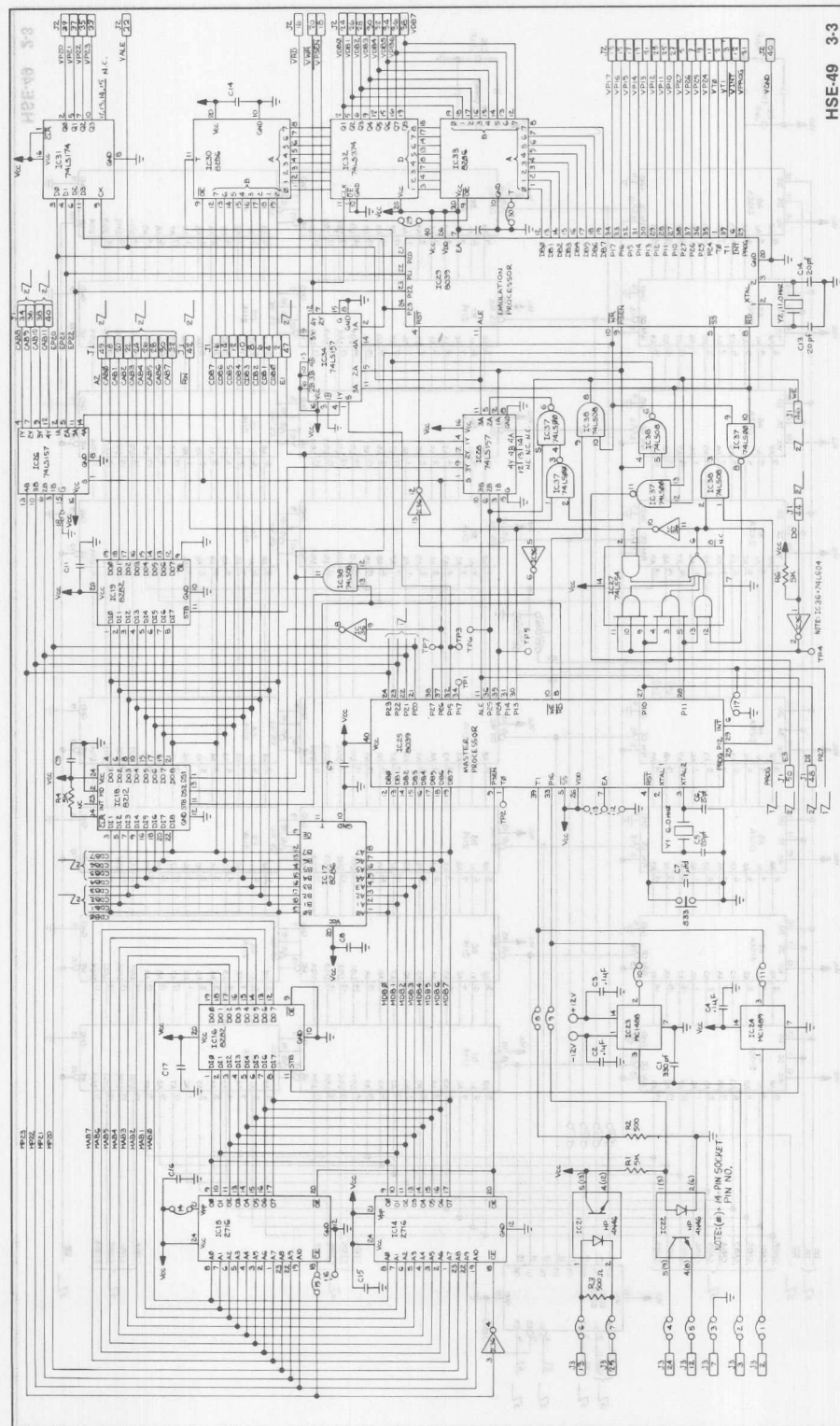
NOTE: POWER SUPPLY IS NOT SHOWN.
 1. APPLICATIONS: POWER SUPPLY (NOT SHOWN)
 2. APPLICATIONS: POWER SUPPLY (NOT SHOWN)
 3. APPLICATIONS: POWER SUPPLY (NOT SHOWN)



Appendix A Schematic Diagrams



Ram Memory



Central Processor


```

LOC 001      LINE      SOURCE STATEMENT
1  MICROPSILE H2E48 MONITOR
2  TITLE('H2E-48(TN) EMULATOR MONITOR VERSION 2.2')
3
4  =====
5  PROGRAM H2E-48(TN) EMULATOR MONITOR
6  VERS 2.2
7
8
9
10  *****
11  *****
12  *****
13  *****
14  *****
15
16  OBJECT
17  =====
18
19  THIS PROGRAM CONTAINS THE SOFTWARE NECESSARY TO RUN THE H2E-48(TN)
20  HIGH-SPEED EMULATOR FOR INTEL'S MCS-48(TN) FAMILY FAMILY OF MICROCOMPUTERS.
21  THE EMULATOR PROVIDES AN ASSORTMENT OF UTILITY FUNCTIONS FOR
22  DEVELOPING AND DEBUGGING 8085-BASED APPLICATIONS, INCLUDING THE
23  ABILITY TO ENTER AND MODIFY PROGRAMS IN PROGRAM RAM.
24  AFTER DATA, SINGLE-STEP, SETTING OF A PROGRAM, AND EXECUTE PROGRAM
25  AT SPEEDS OF UP TO 11 Mhz. WITH OR WITHOUT BREAKPOINTS ENABLED.
26  THE EMULATOR IS DESCRIBED IN GREATER DETAIL IN INTEL'S APPLICATION NOTE
27  0P-85, "A HIGH-SPEED EMULATOR FOR INTEL MCS-48(TN) MICROCOMPUTERS."
28
29  PROGRAM ORGANIZATION
30  =====
31
32  THIS LISTING IS ORGANIZED AS FOLLOWS:
33
34  INTRODUCTION AND HARDWARE OVERVIEW
35  VARIABLE DECLARATION AND DEFINITION
36  POWER-ON SYSTEM INITIALIZATION
37  KEYBOARD COMMAND PARSER AND ASSOCIATED TABLES
38  IMPLEMENTATIONS OF THE PRIMARY COMMANDS
39  WITH ACCESSING UTILITY SUBROUTINES USED THROUGHOUT
40  KEYBOARD COMMANDING AND DISPLAY DRIVING SUBROUTINES
41  KEYBOARD AND DISPLAY INTERFACING UTILITIES
42  ROUTINES AND UTILITY SUBROUTINES WHICH INTERACT BETWEEN MP AND EP.
43
44
45 OBJECT

```

Appendix B Monitor Listings

LOC OBJ LINE SOURCE STATEMENT

```

1 $MACROFILE NOGEN NOCOND XREF
2 $TITLE('HSE-49(TM) EMULATOR MONITOR VERSION 2.5')
3 ;
4 ;*****
5 ;
6 ;          PROGRAM: HSE-49(TM) EMULATOR MONITOR
7 ;          VERS 2.5/709
8 ;
9 ;          COPYRIGHT (C) 1979
10 ;         INTEL CORPORATION
11 ;         3065 BOWERS AVENUE
12 ;         SANTA CLARA, CALIFORNIA 95051
13 ;
14 ;*****
15 ;
16 ; ABSTRACT
17 ; =====
18 ;
19 ; THIS PROGRAM CONTAINS THE SOFTWARE NECESSARY TO RUN THE HSE-49(TM)
20 ; HIGH-SPEED EMULATOR FOR INTEL'S MCS-48(TM) FAMILY FAMILY OF MICROCOMPUTERS.
21 ; THE EMULATOR PROVIDES AN ASSORTMENT OF UTILITY FUNCTIONS FOR
22 ; DEVELOPING AND DEBUGGING 8048-BASED APPLICATIONS, INCLUDING THE
23 ; ABILITY TO ENTER AND MODIFY PROGRAMS IN PROGRAM RAM,
24 ; ALTER DATA, SINGLE-STEP SECTIONS OF A PROGRAM, AND EXECUTE PROGRAMS
25 ; AT SPEEDS OF UP TO 11 MHZ, WITH OR WITHOUT BREAKPOINTS ENABLED.
26 ; THE EMULATOR IS DESCRIBED IN GREATER DEPTH IN INTEL'S APPLICATION NOTE
27 ; AP-55, "A HIGH-SPEED EMULATOR FOR INTEL MCS-48(TM) MICROCOMPUTERS."
28 ;
29 ; PROGRAM ORGANIZATION
30 ; =====
31 ;
32 ; THIS LISTING IS ORGANIZED AS FOLLOWS:
33 ;
34 ;     INTRODUCTION AND HARDWARE OVERVIEW;
35 ;     VARIABLE DECLARATION AND DEFINITION;
36 ;     POWER-ON SYSTEM INITIALIZATION;
37 ;     KEYBOARD COMMAND PARSER AND ASSOCIATED TABLES;
38 ;     IMPLEMENTATIONS OF THE PRIMARY COMMANDS;
39 ;     DATA ACCESSING UTILITY SUBROUTINES USED THROUGHOUT;
40 ;     KEYBOARD SCANNING AND DISPLAY DRIVING SUBROUTINE;
41 ;     KEYBOARD AND DISPLAY INTERFACING UTILITIES;
42 ;     ROUTINES AND UTILITY SUBROUTINES WHICH INTERACT BETWEEN MP AND EP.
43 ;
44 ;
45 $EJECT

```

LOC	OBJ	LINE	SOURCE STATEMENT	THE STATEMENT	LINE	LOC	OBJ
46			INTRODUCTION AND HARDWARE OVERVIEW				
47			=====				
48							
49			THE EMULATOR DESIGN USES TWO MICROPROCESSORS. ONE PROCESSOR CONTROLS				
50			SYSTEM STATUS, INTERPRETS MONITOR COMMANDS, AND COMMUNICATES				
51			WITH THE OUTSIDE WORLD THROUGH THE ON-BOARD KEYBOARD, DISPLAY, SERIAL				
52			INTERFACES, CONTROL SIGNALS, ETC.				
53			A SECOND PROCESSOR IS USED TO ACTUALLY				
54			EXECUTE THE USER'S PROGRAM UNDER THE CONTROL OF THE FIRST.				
55			THESE PROCESSORS ARE REFERRED TO AS THE MASTER PROCESSOR (MP) AND EXECUTION				
56			PROCESSOR (EP) RESPECTIVELY.				
57							
58			THE PROGRAM IN THIS LISTING IS EXECUTED BY THE MASTER PROCESSOR.				
59			AT THE END OF THIS LISTING ARE SEVERAL SHORT "MINI-MONITOR OVERLAYS"				
60			WHICH THE EXECUTION PROCESSOR EXECUTES WHEN INTERACTION BETWEEN THE				
61			TWO PROCESSORS IS NECESSARY.				
62							
63			THIS PROGRAM WAS WRITTEN USING A NUMBER OF MACROS TO HANDLE THE ALLOCATION				
64			OF MPU RESOURCES (WORKING REGISTERS, INTERNAL RAM, AND MP MONITOR ROM)				
65			FOR CODE AND DATA STORAGE. THESE MACRO DEFINITIONS ARE INCLUDED IN A FILE				
66			NAMED "ALLOC.MAC." AND ARE PRINTED IN THIS LISTING FOR REFERENCE.				
67			ANOTHER SET OF MACROS IS USED TO SIMPLIFY THE ACCESSING OF VARIABLES				
68			STORED IN INTERNAL RAM (AS OPPOSED TO WORKING REGISTERS) BY USING R1 TO				
69			INDIRECTLY ADDRESS THE APPROPRIATE RAM LOCATION WHEN NECESSARY.				
70			THESE MACROS ARE INCLUDED IN "MOPCOD.MAC", AND ARE ALSO PRINTED HERE.				
71			COMPLETE UNDERSTANDING OF THESE MACROS IS NOT REQUIRED TO UNDERSTAND THE				
72			MONITOR PROGRAM. ALL LINES WHICH ACTUALLY PRODUCE OBJECT CODE APPEAR IN				
73			THE LISTING ITSELF, INDENTED TWO SPACES FROM THE NORMAL TABULATION COLUMNS.				
74			THE ACTUAL MONITOR PROGRAM FOR THE EMULATOR BEGINS AT APPROXIMATELY				
75			SOURCE LINE NUMBER 500.				
76							
77			LINE GENERATED BY MACRO EXPANSION ARE FLAGGED BY A PLUS SIGN ("+")				
78			IMMEDIATELY FOLLOWING THE SOURCE LINE NUMBER.				
79			A NUMBER OF LINES FROM THE VARIOUS MACRO DEFINITIONS WHICH DO NOT				
80			PRODUCE ANY OBJECT CODE ARE PROCESSED BY THE ASSEMBLER				
81			AS THESE MACROS ARE EXPANDED. WHEN THIS IS THE CASE, THESE LINES ARE				
82			SUPPRESSED FROM THE LIST FILE. AS A RESULT, THE LINE NUMBERS ARE				
83			NOT ALWAYS CONSECUTIVE WHERE A MACRO IS BEING INVOKED.				
84							
85			NOTE:				
86			=====				
87			"SOURCE-LINE" REFERS TO THE DECIMAL NUMBERS LEFT OF EACH INSTRUCTION.				
88			AT THE END OF THE LISTING IS AN ASSEMBLY CROSS-REFERENCE TABLE INDICATING				
89			THE SEQUENTIAL SOURCE-LINE NUMBER OF ALL INSTANCES WHERE ANY VARIABLE				
90			IS DEFINED OR REFERENCED. THIS WILL BE OF GREAT ASSISTANCE IN				
91			LOCATING SPECIFIC SUBROUTINES, ETC. IN THE LISTING.				
92							
93			MNEMONICS COPYRIGHT (C) 1976 INTEL CORPORATION				
94							
95			\$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	STATEMENT	LINE	LOC
		98 \$	INCLUDE(:F0:ALLOC.MAC)		42	
0000		= 99 ?R1	SET 0	INTRODUCTION AND HARDWARE OVERVIEW	43	
		= 100 ;			44	
0000		= 101 ?R00	EQU 0		45	
0001		= 102 ?R01	EQU 1	THE EMULATOR DESIGN USES TWO MICROPROCESSORS	46	
0002		= 103 ?RAM	EQU 2	SYSTEM STATUS, INTERPRETS MONITOR COMMANDS, AND CONTROLS	47	
0003		= 104 ?CONST	EQU 3	WITH THE OUTSIDE WORLD THROUGH THE ON-BOARD KEYBOARD	48	
0004		= 105 ?A	EQU 4 ; ACCUMULATOR VARIABLE TYPE	INTERFACES, CONTROLS	49	
		= 106 ;		A SECOND PROCESSOR IS USED TO ACTUALLY	50	
		= 107 ;	THE FOLLOWING INITIALIZES THE LINKED LIST POINTERS FOR	EXECUTE THE	51	
		= 108 ;	THE REGISTER ALLOCATION AND DEALLOCATION ROUTINES	THESE PROCESSORS	52	
		= 109 ;		THROUGHOUT THIS PROGRAM AS THE HOST PROCESSOR (CP) AND EXECUTION	53	
0003		= 110 ?B0R2	SET 3	PROCESSOR (CP) RESPECTIVELY	54	
0004		= 111 ?B0R3	SET 4		55	
0005		= 112 ?B0R4	SET 5	THE PROGRAM IN THIS LISTING IS EXECUTED BY THE HOST	56	
0006		= 113 ?B0R5	SET 6	AT THE END OF THIS LISTING ARE SEVERAL SHORT MINI-PROGRAMS	57	
0007		= 114 ?B0R6	SET 7	WHICH THE EXECUTION PROCESSOR EXECUTES WHEN INTERVIEW	58	
0008		= 115 ?B0R7	SET 8	TWO PROCESSORS IS NECESSARY	59	
		= 116 ;			60	
0002		= 117 ?B0PNT	SET 2	THIS PROGRAM HAS WRITTEN USING A NUMBER OF MACROS	61	
		= 118 ;		OF NEW RESOURCES WORKING REGISTERS, INTERNAL RAM, AND A MONITOR	62	
0003		= 119 ?B1R2	SET 3	FOR CODE AND DATA STORAGE. THESE MACRO DEFINITION ARE	63	
0004		= 120 ?B1R3	SET 4	NAMED "ALOC.MAC", AND ARE PRINTED IN THIS LISTING FOR REFERENCE	64	
0005		= 121 ?B1R4	SET 5	ANOTHER SET OF MACROS IS USED TO SIMPLY THE ACCESS OF	65	
0006		= 122 ?B1R5	SET 6	STORED IN INTERNAL RAM (AS OPPOSED TO WORKING REGISTERS)	66	
0007		= 123 ?B1R6	SET 7	INDIRECTLY ADDRESS THE APPROPRIATE RAM LOCATION WHERE	67	
0008		= 124 ?B1R7	SET 8	THESE MACROS ARE INCLUDED IN "MONCOP.MAC", AND ARE ALSO	68	
		= 125 ;		COMPLETE UNDERSTANDING OF THESE MACROS IS NOT REQUIRED TO UNDERSTAND	69	
0002		= 126 ?B1PNT	SET 2	MONITOR CONTROL. ALL LINES WHICH ACTUALLY PRODUCE OBJECT	70	
		= 127 ;		THE LISTING ITSELF. INDENTED TWO SPACES FROM THE NORMAL INDENTATION	71	
0000		= 128 ORPG0	SET 000H	THE ACTUAL MONITOR PROGRAM FOR THE EMULATOR DESIGN	72	
0100		= 129 ORPG1	SET 100H	SOURCE LINE NUMBER 500	73	
0200		= 130 ORPG2	SET 200H		74	
0300		= 131 ORPG3	SET 300H	LINES GENERATED BY MACRO EXPANSION ARE FLAGGED	75	
0400		= 132 ORPG4	SET 400H	IMMEDIATELY FOLLOWING THE SOURCE LINE NUMBER	76	
0500		= 133 ORPG5	SET 500H	A NUMBER OF LINES FROM THE VARIOUS MACRO DEFINITIONS	77	
0600		= 134 ORPG6	SET 600H	PRODUCE NEW OBJECT CODE ARE PRECEDED BY THE	78	
0700		= 135 ORPG7	SET 700H	IF THESE MACROS ARE EXPANDED. WHEN THIS IS THE	79	
		= 136 ;		SUPPRESSED FROM THE LIST FILE AS A RESULT. THE LINE NUMBERS ARE	80	
		= 137 \$EJECT		NOT ALWAYS CONSECUTIVE WHEN A MACRO IS BEING INVOKED	81	
				NOTE:	82	
					83	
				"SOURCE-LINE" REFERS TO THE DECIMAL NUMBERS LISTED OF EACH INSTRUCTION	84	
				AT THE END OF THE LISTING IS AN ALPHABETICALLY CROSSED-REFERENCE TABLE INDICATING	85	
				THE SCHEMATIC SOURCE-LINE NUMBER OF ALL INSTRUCTIONS WHERE ANY VARIABLE	86	
				IS DEFINED OR REFERENCED. THIS WILL BE OF GREAT ASSISTANCE IN	87	
				LOCATING SPECIFIC SUBROUTINES, ETC. IN THE LISTING	88	
					89	
				MEMORANDUM COPYRIGHT (C) 1976 INTEL CORPORATION	90	
					91	
					92	
					93	
					94	
					95	
					96	
					97	
					98	
					99	
					100	

LOC	OBJ	LINE	SOURCE STATEMENT	LINE	OBJ
= 138 ;			*****	= 137 ;	
= 139 ;				= 138 ;	
= 140 ;			START OF ALLOCATION MACROS	= 139 ;	
= 141 ;				= 140 ;	
= 142 ;			*****	= 141 ;	
= 143 ;				= 142 ;	
= 144 ?RSAVE			MACRO SYMBOL, BANK, PNTVAL	= 143 ;	
-			PNTVAL EQ 8	= 144 ?RSAVE	
-			ERROR 2	= 145 IF	
-			EXITM	= 146	
-			ENDIF	= 147	
-				= 148	
-			SAVE GEN	= 149 \$	
-			SYMBOL SET R&PNTVAL	= 150	
-			RESTORE	= 151 \$	
-			?B&BANK&PNT SET ?B&BANK&R&PNTVAL	= 152	
-			ENDM	= 153	
-				= 154 ;	
0020				= 155 ;	
-			?MINDX SET 20H	= 156 ?MINDX SET 20H	
-				= 157 ;	
-			?MSAVE MACRO SYMBOL, LENGTH, ADDR	= 158 ?MSAVE MACRO SYMBOL, LENGTH, ADDR	
-			SAVE GEN	= 159 \$	
-			SYMBOL EQU ADDR	= 160	
-			RESTORE	= 161 \$	
-			?MINDX SET ?MINDX+LENGTH	= 162 ?MINDX SET ?MINDX+LENGTH	
-			ENDM	= 163 ENDM	
-				= 164 ;	
-			MBLOCK MACRO SYMBOL, LENGTH	= 165 MBLOCK MACRO SYMBOL, LENGTH	
-			?&SYMBOL EQU 3	= 166 ?&SYMBOL EQU 3	
-			?MSAVE SYMBOL, LENGTH, %MINDX	= 167 ?MSAVE SYMBOL, LENGTH, %MINDX	
-			ENDM	= 168 ENDM	
-				= 169 ;	
-			DECLARE MACRO SYMBOL, TYPE	= 170 DECLARE MACRO SYMBOL, TYPE	
-			?&SYMBOL SET ?&TYPE	= 171 ?&SYMBOL SET ?&TYPE	
-			IF ?&TYPE EQ 2	= 172 IF ?&TYPE EQ 2	
-			?MSAVE SYMBOL, 1, %MINDX	= 173 ?MSAVE SYMBOL, 1, %MINDX	
-			EXITM	= 174 EXITM	
-			ENDIF	= 175 ENDIF	
-			IF ?&TYPE EQ 0	= 176 IF ?&TYPE EQ 0	
-			?RSAVE SYMBOL, 0, %B&PNT	= 177 ?RSAVE SYMBOL, 0, %B&PNT	
-			EXITM	= 178 EXITM	
-			ENDIF	= 179 ENDIF	
-			IF ?&TYPE EQ 1	= 180 IF ?&TYPE EQ 1	
-			?RSAVE SYMBOL, 1, %B&PNT	= 181 ?RSAVE SYMBOL, 1, %B&PNT	
-			EXITM	= 182 EXITM	
-			ENDIF	= 183 ENDIF	
-			ENDM	= 184 ENDM	
-				= 185 ;	
-			EJECT	= 186 \$ EJECT	

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOL TABLE	VALUE	LOC	OBJ
		= 187 ;	*****				
		= 188 ;REORG	MACRO TO RESET THE INSTRUCTION LOCATION COUNTER				
		= 189 ;	TO THE FIRST FREE LOCATION ON THE FIRST PAGE MODULE WILL				
		= 190 ;	FIT WITHIN.				
		= 191 REORG	MACRO LOCATION				
		= 192 \$SAVE GEN					
-		= 193	ORG LOCATION				
-		= 194 \$RESTORE					
		= 195	ENDM				
		= 196 ;					
		= 197 ;CODEBLK	MACRO TO FIND A PAGE OF ROM				
		= 198 ;	WHICH THIS BLOCK OF CODE WILL FIT WITHIN				
		= 199 CODEBLK	MACRO LENGTH				
		= 200 ?LENGTH	SET LENGTH				
-		= 201 IF	HIGH(ORGP0+LENGTH-1) EQ 0				
-		= 202	REORG %ORGP0				
-		= 203 ?START	SET \$				
-		= 204 EXITM					
-		= 205 ENDIF					
-		= 206 IF	HIGH(ORGP1+LENGTH-1) EQ 1				
-		= 207	REORG %ORGP1				
-		= 208 ?START	SET \$				
-		= 209 EXITM					
-		= 210 ENDIF					
-		= 211 IF	HIGH(ORGP2+LENGTH-1) EQ 2				
-		= 212	REORG %ORGP2				
-		= 213 ?START	SET \$				
-		= 214 EXITM					
-		= 215 ENDIF					
-		= 216 IF	HIGH(ORGP4+LENGTH-1) EQ 4				
-		= 217	REORG %ORGP4				
-		= 218 ?START	SET \$				
-		= 219 EXITM					
-		= 220 ENDIF					
-		= 221 IF	HIGH(ORGP5+LENGTH-1) EQ 5				
-		= 222	REORG %ORGP5				
-		= 223 ?START	SET \$				
-		= 224 EXITM					
-		= 225 ENDIF					
-		= 226 IF	HIGH(ORGP6+LENGTH-1) EQ 6				
-		= 227	REORG %ORGP6				
-		= 228 ?START	SET \$				
-		= 229 EXITM					
-		= 230 ENDIF					
-		= 231 IF	HIGH(ORGP7+LENGTH-1) EQ 7				
-		= 232	REORG %ORGP7				
-		= 233 ?START	SET \$				
-		= 234 EXITM					
-		= 235 ENDIF					
-		= 236 IF	HIGH(ORGP3+LENGTH-1) EQ 3				
-		= 237	REORG %ORGP3				
-		= 238 ?START	SET \$				
-		= 239 EXITM					
-		= 240 ENDIF					
-		= 241	ERROR 0 ;*** INSUFFICIENT SPACE FOR CODE ON ANY PAGE ***				

LOC	OBJ	LINE	SOURCE STATEMENT	THIRDTAT2 330002	LINE	LOC 301
= 242		ENDM			300	
= 243	:	DATABLK	INSERTS ONTO PAGE 3	301		
= 244	:	DATABLK MACRO	LENGTH	302		
= 245	?	LENGTH SET	LENGTH	303		
= 246	IF	HIGH(ORGP63+LENGTH-1) EQ 3		304		
= 247		REORG	%ORGP63	305		
= 248	?	START SET	\$	306		
= 249		EXITM		307		
= 250		ENDIF		308		
= 251		ERROR 0	*** INSUFFICIENT SPACE FOR DATA BLOCK ON PAGE 3 ***	309		
= 252		ENDM		310		
= 253	:	SIZE	PRINTS A LINE TO THE SOURCE FILE GIVING BLOCK SIZE	311		
= 254	:		AND UPDATES APPROPRIATE ORGP6#	312		
= 255	?	SIZE MACRO	BLK, PGE	313		
= 256		\$SAVE GEN		314		
= 257		SIZE SET	BLK	315		
= 258	:			316		
= 259	:		*****	317		
= 260	IF	?LENGTH LT SIZE		318		
= 261		ERROR 0	*** SIZE EXCEEDS SPACE CHECKED FOR BY CODEBLK MACRO	319		
= 262		ENDIF		320		
= 263	IF	HIGH(\$-1) NE HIGH(?START)		321		
= 264		ERROR 0	*** CODE OR DATA BLOCK ROLLED OVER PAGE BOUNDARY ***	322		
= 265		ENDIF		323		
= 266		\$RESTORE		324		
= 267		ORGP6#PGE	SET \$	325		
= 268		ENDM		326		
= 269	:	SIZECHK	CHECKS SIZE OF PRECEDING BLOCK, PRINTS SIZE TO .LS1 FILE	327		
= 270	:	SIZECHK MACRO		328		
= 271	?	SIZE	%(\$-?START), %HIGH(?START)	329		
= 272		ENDM		330		
= 273	:			331		
= 274	:			332		
= 275	:	RSOURCE	CODE SPACE ALLOCATION SUMMARY STATEMENT	333		
= 276		RSource MACRO		334		
= 277		\$SAVE LIST GEN		335		
= 278		PGSIZE SET	ORGP60-000H ; BYTES USED ON PAGE 0	336		
= 279		PGSIZE SET	ORGP61-100H ; BYTES USED ON PAGE 1	337		
= 280		PGSIZE SET	ORGP62-200H ; BYTES USED ON PAGE 2	338		
= 281		PGSIZE SET	ORGP63-300H ; BYTES USED ON PAGE 3	339		
= 282		PGSIZE SET	ORGP64-400H ; BYTES USED ON PAGE 4	340		
= 283		PGSIZE SET	ORGP65-500H ; BYTES USED ON PAGE 5	341		
= 284		PGSIZE SET	ORGP66-600H ; BYTES USED ON PAGE 6	342		
= 285		PGSIZE SET	ORGP67-700H ; BYTES USED ON PAGE 7	343		
= 286		\$EJECT		344		
= 287		\$RESTORE		345		
= 288				346		
= 289		\$EJECT		347		

LOC	OBJ	LINE	SOURCE STATEMENT	OBJECT STATEMENT	LINE	LOC	OBJ
		290 ;					
		291 \$	INCLUDE(:F0:MOPCOD.MAC)				
		= 292 ;					
		= 293 ; ?FORM1 MACRO	FOR GENERALIZING OPCODE INSTRUCTION				
		= 294 ;					
		= 295 ?FORM1 MACRO	OPCODE, SRC				
		= 296 IF	?&SRC EQ 2				
		= 297 \$	SAVE GEN				
		= 298	MOV R1, #SRC				
		= 299	OPCODE A, @R1				
		= 300 \$	RESTORE				
		= 301	EXITM				
		= 302 ENDIF					
		= 303 IF	?&SRC EQ 0 OR ?&SRC EQ 1				
		= 304 \$	SAVE GEN				
		= 305	OPCODE A, SRC				
		= 306 \$	RESTORE				
		= 307	EXITM				
		= 308 ENDIF					
		= 309 IF	?&SRC EQ 3				
		= 310 \$	SAVE GEN				
		= 311	OPCODE A, #SRC				
		= 312 \$	RESTORE				
		= 313	EXITM				
		= 314 ENDIF					
		= 315	ERROR 1				
		= 316 ENDM					
		= 317 ;					
		= 318 ; ?FORM2 MACRO	FOR GENERALIZING MOVES FROM THE ACC TO A VARIABLE				
		= 319 ?FORM2 MACRO	DEST				
		= 320 IF	?&DEST EQ 2				
		= 321 \$	SAVE GEN				
		= 322	MOV R1, #DEST				
		= 323	MOV @R1, A				
		= 324 \$	RESTORE				
		= 325	EXITM				
		= 326 ENDIF					
		= 327 IF	?&DEST EQ 0 OR ?&DEST EQ 1				
		= 328 \$	SAVE GEN				
		= 329	MOV @DEST, A				
		= 330 \$	RESTORE				
		= 331	EXITM				
		= 332 ENDIF					
		= 333	ERROR 1				
		= 334 ENDM					
		= 335 ;					
		= 336 ; ?FORM3 MACRO	FOR GENERALIZING MOVES FROM THE ACC TO A VARIABLE				
		= 337 ;	WHEN IT IS KNOWN THAT R1 (IF NEEDED FOR INDIRECT ADDRESSING)				
		= 338 ;	IS ALREADY PRESET.				
		= 339 ?FORM3 MACRO	DEST				
		= 340 IF	?&DEST EQ 2				
		= 341 \$	SAVE GEN				
		= 342	MOV @R1, A				
		= 343 \$	RESTORE				
		= 344	EXITM				

```

= 347 $      SAVE GEN
= 348      MOV      DEST, A
= 349 $      RESTORE
= 350      EXITM
= 351 ENDIF
= 352      ERROR    1
= 353 ENDM
= 354 ;
= 355 ;?FORM4 MACRO  FOR GENERALIZING 'MOV  A, SRC' INSTRUCTION
= 356 ?FORM4 MACRO  SRC
= 357 IF      ?&SRC EQ 2
= 358 $      SAVE GEN
= 359      MOV      R1, #SRC
= 360      MOV      A, @R1
= 361 $      RESTORE
= 362      EXITM
= 363 ENDIF
= 364 IF      ?&SRC EQ 0 OR ?&SRC EQ 1
= 365 $      SAVE GEN
= 366      MOV      A, SRC
= 367 $      RESTORE
= 368      EXITM
= 369 ENDIF
= 370 IF      ?&SRC EQ 3
= 371 $      SAVE GEN
= 372      MOV      A, #SRC
= 373 $      RESTORE
= 374      EXITM
= 375 ENDIF
= 376      ERROR    1
= 377 ENDM
= 378 ;
= 379 ;?FORM5 MACRO  FOR GENERALIZING MOVING A CONSTANT INTO A VARIABLE
= 380 ?FORM5 MACRO  DEST, CONST
= 381 IF      ?&DEST EQ 0 OR ?&DEST EQ 1 OR ?&DEST EQ 4
= 382 $      SAVE GEN
= 383      MOV      DEST, #CONST
= 384 $      RESTORE
= 385      EXITM
= 386 ENDIF
= 387 IF      ?&DEST EQ 2
= 388 $      SAVE GEN
= 389      MOV      R1, #DEST
= 390      MOV      @R1, #CONST
= 391 $      RESTORE
= 392      EXITM
= 393 ENDIF
= 394      ERROR    1
= 395 ENDM
= 396 ;
= 397 ;MMOV MACRO  GENERALIZED MOVE FROM SRC TO DEST
= 398 MMOV MACRO  DEST, SRC
= 399 IF      ?&SRC EQ 3

```

LOC	OBJ	LINE	SOURCE STATEMENT	OBJECT STATEMENT	LOC	OBJ
-		= 404	?FORM1 MOV, SRC	RESTORE	= 394	-
-		= 405	EXITM	EXITM	= 395	-
-		= 406	ENDIF	ENDIF	= 396	-
-		= 407	IF ?&SRC EQ 4	ERROR 1	= 397	-
-		= 408	?FORM2 DEST	MOV	= 398	-
-		= 409	EXITM	EXITM	= 399	-
-		= 410	ENDIF	ENDIF	= 400	-
-		= 411	?FORM1 MOV, SRC	RESTORE	= 401	-
-		= 412	?FORM2 DEST	MOV	= 402	-
-		= 413	ENDM	ENDM	= 403	-
-		= 414	?BINOP MACRO GENERALIZES ARITHMETIC AND LOGICAL OPERATIONS	MACRO	= 404	-
-		= 415	?BINOP MACRO OPCODE, DEST, SRC	MACRO	= 405	-
-		= 416	IF ?&DEST EQ 4	ERROR 1	= 406	-
-		= 417	?FORM1 OPCODE, SRC	RESTORE	= 407	-
-		= 418	EXITM	EXITM	= 408	-
-		= 419	ENDIF	ENDIF	= 409	-
-		= 420	IF ?&SRC EQ 4	ERROR 1	= 410	-
-		= 421	?FORM1 OPCODE, DEST	MOV	= 411	-
-		= 422	?FORM3 DEST	RESTORE	= 412	-
-		= 423	EXITM	EXITM	= 413	-
-		= 424	ENDIF	ENDIF	= 414	-
-		= 425	?FORM1 MOV, SRC	RESTORE	= 415	-
-		= 426	?FORM1 OPCODE, DEST	MOV	= 416	-
-		= 427	?FORM3 DEST	MOV	= 417	-
-		= 428	ENDM	RESTORE	= 418	-
-		= 429	?MADD MACRO FOR GENERALIZING ADD INSTRUCTION	EXITM	= 419	-
-		= 430	?MADD MACRO DEST, SRC	MACRO	= 420	-
-		= 431	?BINOP ADD, DEST, SRC	ERROR 1	= 421	-
-		= 432	ENDM	MACRO	= 422	-
-		= 433	;	;	= 423	-
-		= 434	?MADDC MACRO FOR GENERALIZING ADDC INSTRUCTION	MACRO	= 424	-
-		= 435	?MADDC MACRO DEST, SRC	MACRO	= 425	-
-		= 436	?BINOP ADDC, DEST, SRC	ERROR 1	= 426	-
-		= 437	ENDM	MACRO	= 427	-
-		= 438	;	;	= 428	-
-		= 439	?MANL MACRO FOR GENERALIZING ANL INSTRUCTION	RESTORE	= 429	-
-		= 440	?MANL MACRO DEST, SRC	EXITM	= 430	-
-		= 441	?BINOP ANL, DEST, SRC	ERROR 1	= 431	-
-		= 442	ENDM	MACRO	= 432	-
-		= 443	;	;	= 433	-
-		= 444	?MORL MACRO FOR GENERALIZING ORL INSTRUCTION	RESTORE	= 434	-
-		= 445	?MORL MACRO DEST, SRC	MOV	= 435	-
-		= 446	?BINOP ORL, DEST, SRC	RESTORE	= 436	-
-		= 447	ENDM	EXITM	= 437	-
-		= 448	;	;	= 438	-
-		= 449	?MXRL MACRO FOR GENERALIZING XRL INSTRUCTION	ERROR 1	= 439	-
-		= 450	?MXRL MACRO DEST, SRC	MACRO	= 440	-
-		= 451	?BINOP XRL, DEST, SRC	MACRO	= 441	-
-		= 452	ENDM	MACRO	= 442	-
-		= 453	;	;	= 443	-
-		= 454	?MXCH MACRO FOR GENERALIZING XCH INSTRUCTION	ERROR 1	= 444	-

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOL TABLE	LINE	LOC	OBJ
		= 455	MXCH MACRO DEST, SRC		200		
-		= 456	?BINOP XCH, DEST, SRC		200		
		= 457	ENDM		200		
		= 458 ;			200		
		= 459 ?UNARY MACRO OPCODE, DEST		200			
-		= 460 ?FORM1 MOV, DEST		200			
-		= 461 \$SAVE GEN		200			
-		= 462 OPCODE A		200			
-		= 463 \$RESTORE		200			
-		= 464 ?FORM3 DEST		200			
		= 465 ENDM		200			
		= 466 ;		200			
		= 467 MINC MACRO DEST		200			
-		= 468 ?UNARY INC, DEST		200			
		= 469 ENDM		200			
		= 470 ;		200			
		= 471 MDEC MACRO DEST, ADDR		200			
-		= 472 ?UNARY DEC, DEST		200			
		= 473 ENDM		200			
		= 474 ;		200			
		= 475 MDJNZ MACRO DEST, ADDR		200			
-		= 476 ?UNARY DEC, DEST		200			
-		= 477 \$SAVE GEN		200			
-		= 478 JNZ ADDR		200			
-		= 479 \$RESTORE		200			
		= 480 ENDM		200			
		= 481 ;		200			
		= 482 MRL MACRO DEST		200			
-		= 483 ?UNARY RL, DEST		200			
		= 484 ENDM		200			
		= 485 ;		200			
		= 486 MRR MACRO DEST		200			
-		= 487 ?UNARY RR, DEST		200			
		= 488 ENDM		200			
		= 489 ;		200			
		= 490 MRRC MACRO DEST		200			
-		= 491 ?UNARY RRC, DEST		200			
		= 492 ENDM		200			
		= 493 ;		200			
		= 494 MRLC MACRO DEST		200			
-		= 495 ?UNARY RLC, DEST		200			
		= 496 ENDM		200			
		= 497 ;		200			
		= 498 \$EJECT		200			

LOC	OBJ	LINE	SOURCE STATEMENT	INSTRUMENT	TIME	DATE
		499 ;				
		500 ;	=====			
		501 ;	=====			
		502 ;	BEGINNING OF PROGRAM PROPER			
		503 ;	=====			
		504 ;	=====			
		505 ;				
		506 ;				
		507 ;	*****			
		508 ;				
		509 ;	ALLOCATION OF MP I/O PORTS:			
		510 ;				
		511 ;	*****			
		512 ;				
		513 ;	BUS ; USED FOR BIDIRECTIONAL ADDRESS AND DATA TRANSFERS			
		514 ;	P1 ; USED AS INDIVIDUAL CONTROL OUTPUTS AND BREAK LOGIC			
		515 ;	P2 ; HIGH-ORDER ADDRESS AND ADDRESS SPACE SELECTION			
		516 ;				
000E		517 PDIGIT EQU P7	; USED TO ENABLE CHARACTERS AND STROBE ROWS OF KEYBOARD			
000D		518 PSEGHI EQU P6	; USED TO TURN ON HI SEGMENTS OF CURRENTLY ENABLED DIGIT			
000C		519 PSEGLO EQU P5	; PORT FOR LOWER FOUR SEGMENTS			
000B		520 PINPUT EQU P4	; PORT USED TO SCAN FOR KEY CLOSURES			
		521 ;				
		522 ;	*****			
		523 ;				
		524 ;	INDIVIDUAL PINS OF PORT 1 USED AS FOLLOWS:			
		525 ;				
		526 ;	*****			
		527 ;				
0001		528 ENDRAM EQU 0000001B	; P10 - HI ENABLES BREAK ON BREAK RAM OUTPUT SIGNAL			
0002		529 ENBLNK EQU 00000010B	; P11 - HI ENABLES BREAK ON RD OR WR TO LINK BY EP			
		530	; (NOTE: P11 & P10 BOTH HI ENABLES			
		531	; BREAK ON ANY EP INSTRUCTION CYCLE)			
0004		532 EPSSTP EQU 00000100B	; P12 - LO FORCES EP SS INPUT LOW			
		533	; HI GATES BREAKPOINT FLIP-FLOP TO EP SS INPUT.			
0008		534 CLRBF EQU 00001000B	; P13 - LO CLEARS BREAK FLIP-FLOP			
		535	; AND ENABLES WR CONTROL TO BREAKPOINT RAM.			
0010		536 EPRSET EQU 00010000B	; P14 - HI RESETS EP			
0020		537 MODOUT EQU 00100000B	; P15 - LO WHEN EP IS EXECUTING USER PROGRAM			
		538	; HI WHEN EP FROZEN OR RUNNING OVERLAYS.			
0040		539 ITYOUT EQU 01000000B	; P16 - SERIAL OUTPUT TO TTY OR CRT			
		540	; P17 - UNUSED			
		541 ;				
		542 \$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	ADDRESS	VALUE	TYPE
		543 ;	*****			
		544 ;	*****			
		545 ;	INDIVIDUAL PINS OF PORT 2 USED AS FOLLOWS:			
		546 ;	*****			
		547 ;	*****			
		548 ;	*****			
		549 ;	P23-P20 ; ADR11-ADR8 FOR ACCESSING PROGRAM OR DATA RAM ARRAY			
		550 ;	*****			
0010		551 M0	EQU 00010000B ; P24 -- MEMORY MATRIX CONTROL PIN 0			0000
0020		552 M1	EQU 00100000B ; P25 -- MEMORY MATRIX CONTROL PIN 1			
0040		553 MPUSEL	EQU 01000000B ; P26 -- HIGH WHEN MP IN CONTROL OF COMMON MEM ARRAY,			
		554	LOW WHEN EP IN CONTROL			
0080		555 EXPMON	EQU 10000000B ; P27 -- JUMPED TO GROUND FOR STANDARD MONITOR,			
		556	FLOATING WHEN EXPANSION MONITOR PRESENT.			
		557 ;	*****			
		558 ;	*****			
		559 ;	WHEN MP IN CONTROL OF MEMORY MATRIX M1-M0 USED AS FOLLOWS:			
		560 ;	*****			
		561 ;	M1 M0 MODE			
		562 ;	0 0 PROGRAM RAM ARRAY ENABLED FOR READ & WRITE			
		563 ;	0 1 DATA RAM ARRAY ENABLED FOR READ & WRITE			
		564 ;	1 X LINK REGISTER ENABLED FOR READ, RAM ARRAYS DISABLED.			
		565 ;	***** (NOTE: LINK REGISTER ALWAYS ENABLED FOR MP WRITES) *****			
		566 ;	*****			
		567 ;	WHEN EP IN CONTROL OF MATRIX M1-M0 USED AS FOLLOWS:			
		568 ;	*****			
		569 ;	M1 M0 MODE			
		570 ;	0 X EP PSEN FETCHES FROM LINK REGISTER (USED TO FORCE OPCODES)			
		571 ;	1 0 EP PSEN FETCHES FROM PROGRAM RAM ARRAY,			
		572 ;	RD & WR CONTROL DATA RAM ARRAY.			
		573 ;	1 1 EP PSEN FETCHES FROM PROGRAM RAM ARRAY,			
		574 ;	RD & WR CONTROL LINK REGISTER.			
		575 ;	*****			
		576	#EJECT			

```

580 ;      SYSTEM CONSTANT DEFINITIONS:
581 ;
582 ;*****
583 ; *****
584 DECLARE CHARN0,CONST ;NUMBER OF DIGITS IN DISPLAY AND ROWS OF KEYS
0008      CHARN0 EQU 8
588      CHARN0 EQU 8
589 ; *****
600 DECLARE NCOLS,CONST ;LESSER DIMENSION OF KEYBOARD MATRIX
0004      NCOLS EQU 4
614      NCOLS EQU 4
615 ; *****
616 DECLARE DEBNCE,CONST ;NUMBER OF SUCCESSIVE SCANS BEFORE KEY CLOSURE ACCEPTED
0008      DEBNCE EQU 8
630      DEBNCE EQU 8
631 ; *****
632 DECLARE OVSZ0,CONST ;SIZE OF LARGEST MINI-MONITOR OVERLAY FOR EIP
0017      OVSZ0 EQU 23
646      OVSZ0 EQU 23
647 ; *****
648 DECLARE BUFLN,CONST ;LENGTH OF HEX FORMAT XMIT BUFFER (MAX RECORD LENGTH)
0010      BUFLN EQU 16
662      BUFLN EQU 16
663 ; *****
664 ;*****
665 ; *****
666 ;      UTILITY CONSTANT DECLARATIONS
667 ; *****
668 ;*****
669 ;*****
670 DECLARE ZERO,CONST
0000      ZERO EQU 0
684      ZERO EQU 0
685 DECLARE PLUS1,CONST
0001      PLUS1 EQU 1
699      PLUS1 EQU 1
700 DECLARE PLUS3,CONST
0003      PLUS3 EQU 3
714      PLUS3 EQU 3
715 DECLARE NEG1,CONST
FFFF      NEG1 EQU -1
729      NEG1 EQU -1
730 ; *****
731 $EJECT

```

	735 ;	BANK 0 REGISTER ALLOCATION:	
	736 ;		
	737 ;	*****	
	738 ;		
0002	739	DECLARE LDATA,RB0 ; DATA USED BY LOGICAL ADDRESSING READ/WRITE UTILITIES	
	752+	LDATA SET R2	
	756	DECLARE KEY,RB0 ; HOLDS KEYCODE RETURNED FROM KBD INPUT ROUTINE	
0003	769+	KEY SET R3	
	773	DECLARE ITMP,RB0 ; COUNTER USED AS AN INDEX IN PARSER ROUTINE	
0004	786+	ITMP SET R4	
	790	DECLARE CHKSUM,RB0 ; CHECKSUM OF DATA BYTES TRANSMITTED IN HEX FILE FORMAT	
0005	803+	CHKSUM SET R5	
	807	DECLARE DSPTMP,RB0 ; TEMPORARY STORAGE FOR DISPLAY PATTERNS IN 'DSPACC'	
0006	820+	DSPTMP SET R6	
	824	DECLARE XPCODE,RB0 ; EXPANSION MONITOR ROUTINE CODE NUMBER	
0007	837+	XPCODE SET R7	
	841 ;		
	842 ;	*****	
	843 ;		
	844 ;	BANK 1 REGISTER ALLOCATION	
	845 ;		
	846 ;	*****	
	847 ;		
	848	DECLARE ROTPAT,RB1 ; USED TO HOLD INPUT PATTERN BEING ROTATED THROUGH CV	
0002	865+	ROTPAT SET R2	
	869	DECLARE ROTCNT,RB1 ; COUNTS NUMBER OF BITS ROTATED THROUGH CV	
0003	886+	ROTCNT SET R3	
	890	DECLARE LASTKY,RB1 ; HOLDS KEY POSITION OF LAST KEY DEPRESSION DETECTED	
0004	907+	LASTKY SET R4	
	911	DECLARE CURDIG,RB1 ; HOLDS POSITION OF NEXT CHARACTER TO BE DISPLAYED	
0005	928+	CURDIG SET R5	
	932	DECLARE KEYFLG,RB1 ; FLAG TO DETECT WHEN ALL KEYS ARE RELEASED	
0006	949+	KEYFLG SET R6	
	953	; (REGISTER 7 NOT USED FOR PRIMARY MONITOR)	
	954 ;		
	955 ;	*****	
	956	\$EJECT	

LOC	OBJ	LINE	SOURCE STATEMENT	LINE	LOC
		957 ;			
		958 ;*****			
		959 ;			
		960 ; DATA RAM ALLOCATION			
		961 ;			
		962 ;*****			
		963 ;			
0020		964 DECLARE EPACC, RAM ; STORAGE IN MP FOR EP ACCUMULATOR			
		969+ EPACC EQU 32			
0021		973 DECLARE EPPSW, RAM ; STORAGE IN MP FOR EP PROGRAM STATUS WORD			
		978+ EPPSW EQU 33			
0022		982 DECLARE EPTMR, RAM ; STORAGE IN MP FOR EP TIMER/COUNTER REGISTER			
		987+ EPTMR EQU 34			
0023		991 DECLARE EPR0, RAM ; STORAGE IN MP FOR EP REGISTER 0 OF BANK 0			
		996+ EPR0 EQU 35			
0024		1000 DECLARE EPPCLO, RAM ; STORAGE IN MP FOR LOW BYTE OF EP PROGRAM COUNTER			
		1005+ EPPCLO EQU 36			
0025		1009 DECLARE EPPCHI, RAM ; STORAGE IN MP FOR HIGH NIBBLE OF EP PROGRAM COUNTER			
		1014+ EPPCHI EQU 37			
0026		1018 DECLARE HBITLO, RAM ; PARAMETER 1 FOR SERIAL LINK DATA RATE GENERATOR			
		1023+ HBITLO EQU 38			
0027		1027 DECLARE HBITHI, RAM ; PARAMETER 2 FOR SERIAL LINK DATA RATE GENERATOR			
		1032+ HBITHI EQU 39			
0028		1036 DECLARE DSPTIM, RAM ; PARAMETER FOR AUTO-STEP AND AUTO-BREAK SEQUENCING RATE			
		1041+ DSPTIM EQU 40			
0029		1045 DECLARE VERSNO, RAM ; MONITOR VERSION NUMBER			
		1050+ VERSNO EQU 41			
002A		1054 DECLARE HREGA, RAM ; (UNUSED)			
		1059+ HREGA EQU 42			
002B		1063 DECLARE HREGB, RAM ; (UNUSED)			
		1068+ HREGB EQU 43			
002C		1072 DECLARE HREGC, RAM ; (UNUSED)			
		1077+ HREGC EQU 44			
002D		1081 DECLARE HREGD, RAM ; (UNUSED)			
		1086+ HREGD EQU 45			
002E		1090 DECLARE HREGF, RAM ; (UNUSED)			
		1095+ HREGF EQU 46			
002F		1099 DECLARE HREGF, RAM ; (UNUSED)			
		1104+ HREGF EQU 47			
0030		1108 DECLARE SMALO, RAM ; PRIMARY COMMAND STARTING MEMORY ADDRESS (LOW BYTE)			
		1113+ SMALO EQU 48			
0031		1117 DECLARE SMAHI, RAM ; PRIMARY COMMAND STARTING MEMORY ADDRESS (HIGH BYTE)			
		1122+ SMAHI EQU 49			
0032		1126 DECLARE EMALO, RAM ; PRIMARY COMMAND ENDING MEMORY ADDRESS (LOW BYTE)			
		1131+ EMALO EQU 50			
0033		1135 DECLARE EMAHI, RAM ; PRIMARY COMMAND ENDING MEMORY ADDRESS (HIGH BYTE)			
		1140+ EMAHI EQU 51			
0034		1144 DECLARE MEMLO, RAM ; THIRD PARSER PARAMETER & HEX RECORD ADDRESS (LOW)			
		1149+ MEMLO EQU 52			
0035		1153 DECLARE MEMHI, RAM ; THIRD PARSER PARAMETER & HEX RECORD ADDRESS (HIGH)			
		1158+ MEMHI EQU 53			
0036		1162 DECLARE BCODE, RAM ; PRIMARY COMMAND NUMBER FROM PARSER TABLES (0-8)			
		1167+ BCODE EQU 54			
0037		1171 DECLARE TYPE, RAM ; PRIMARY COMMAND MODIFIER/OPTION (0-5)			
		1176+ TYPE EQU 55			

LOC	OBJ	LINE	SOURCE STATEMENT	THIRTYTHREE 339002	LINE	180	30J
		1180	DECLARE NUMCON, RAM ; MAX. NUMBER OF PARAMETERS ALLOWED FOR SELECTED COMMAND				
0038		1185+	NUMCON EQU 56				
		1189	DECLARE OPTION, RAM ; INDEX POINTER USED IN SEARCHING PARSER TABLES				
0039		1194+	OPTION EQU 57				
		1198	DECLARE NEXTPL, RAM ; CHARACTER POSITION FOR DISPLAY UTILITIES TO WRITE NEXT				
003A		1203+	NEXTPL EQU 58				
		1207	DECLARE KBDBUF, RAM ; POSITION OF KEY DEBOUNCE BY SCANNING SUBROUTINE				
003B		1212+	KBDBUF EQU 59				
		1216	DECLARE KEYLOC, RAM ; INCREMENTED AS SUCCESSIVE KEY LOCATIONS SCANNED				
003C		1221+	KEYLOC EQU 60				
		1225	DECLARE NREPTS, RAM ; KEEPS TRACK OF SUCCESSIVE READS OF SAME KEYSTROKE				
003D		1230+	NREPTS EQU 61				
		1234	DECLARE ASAVE, RAM ; HOLDS ACCUMULATOR VALUE DURING SERVICE ROUTINE				
003E		1239+	ASAVE EQU 62				
		1243	DECLARE RDELAY, RAM ; COUNTER DECREMENTED WHEN AUTO-STEP DELAY IN PROGRESS				
003F		1248+	RDELAY EQU 63				
		1252	DECLARE STRTMP, RAM ; INDEX POINTER FOR DISPLAY CHARACTER STRING ACCESSING				
0040		1257+	STRTMP EQU 64				
		1261	DECLARE BUFCNT, RAM ; COUNT OF DATA BYTES IN HEX FORMAT RECORD BUFFER				
0041		1266+	BUFCNT EQU 65				
		1270	DECLARE RECTYP, RAM ; TYPE OF HEX FORMAT RECORD (0 OR 1)				
0042		1275+	RECTYP EQU 66				
		1279	DECLARE B, RAM ; BIT COUNTER FOR ASCII SERIAL I/O UTILITY SUBROUTINES				
0043		1284+	B EQU 67				
		1288	DECLARE REGC, RAM ; CHARACTER BEING SHIFTED DURING SERIAL I/O PROCESS				
0044		1293+	REGC EQU 68				
		1297	DECLARE H, RAM ; COUNTER IN SOFTWARE DELAY DATA RATE GENERATOR				
0045		1302+	H EQU 69				
		1306 ;					
		1307	MBLOCK SEGMAP, CHARN0 ; REGISTER ARRAY FOR DISPLAY PATTERNS				
0046		1311+	SEGMAP EQU 70				
		1314 ;					
		1315	MBLOCK OYBUF, OYSIZE ; LOW-ORDER USER PROGRAM DURING MINI-MONITOR OVERLAYS				
004E		1319+	OYBUF EQU 78				
		1322 ;					
		1323	MBLOCK HEXBUF, BUFLN ; ALLOCATE BLOCK OF RAM FOR USE AS HEX RECORD BUFFER				
0065		1327+	HEXBUF EQU 101				
		1330 ;					
		1331	\$EJECT				

```

1341 ; INVALS TABLE OF CONSTANTS TO BE LOADED INTO MP INTERNAL RAM VARIABLES
1342 ; AS PART OF SYSTEM INITIALIZATION PROCEDURE:
1343 ;
1344 ;          INITIAL VALUE  VARIABLE      TYPE
1345 ;          =====
0300 00 1346 INVALS: DB      00H      ; ROTPAT      RB1
0301 00 1347          DB      00H      ; ROTCNT      RB1
0302 00 1348          DB      00H      ; LASTKY      RB1
0303 00 1349          DB      CHARNO    ; CURDIG      RB1
0304 00 1350          DB      00H      ; KEYFLG      RB1
0305 00 1351          DB      00H      ; <REG7>      RB1
0306 00 1352          DB      00H      ; EPACC      RAM
0307 01 1353          DB      01H      ; EPPSW      RAM
0308 00 1354          DB      00H      ; EPTIMR     RAM
0309 00 1355          DB      00H      ; EPR0       RAM
030A 00 1356          DB      00H      ; EPPCLO     RAM
030B 00 1357          DB      00H      ; EPPCHI     RAM
030C 93 1358          DB      93H      ; HBITLO     RAM
030D 04 1359          DB      04H      ; HBITHI     RAM
030E 20 1360          DB      20H      ; DSPTIM     RAM
030F 25 1361          DB      25H      ; VERSNO     RAM
0310 00 1362          DB      00H      ; HREGA      RAM
0311 00 1363          DB      00H      ; HREGB      RAM
0312 00 1364          DB      00H      ; HREGC      RAM
0313 00 1365          DB      00H      ; HREGD      RAM
0314 00 1366          DB      00H      ; HREG E      RAM
0315 00 1367          DB      00H      ; HREGF      RAM
0316 00 1368          DB      00H      ; SMALO      RAM
0317 00 1369          DB      00H      ; SMAHI      RAM
0318 FF 1370          DB      0FFH     ; EMAILO     RAM
0319 0F 1371          DB      0FH      ; EMAHI      RAM
031A 00 1372          DB      00H      ; MEMLO      RAM
031B 00 1373          DB      00H      ; MEMHI      RAM
031C 00 1374          DB      00H      ; BCODE      RAM
031D 04 1375          DB      04H      ; TYPE       RAM
031E 01 1376          DB      01H      ; NUMCON     RAM
031F 00 1377          DB      00H      ; OPTION     RAM
0320 00 1378          DB      CHARNO    ; NEXTPL     RAM
0321 FF 1379          DB      0FFH     ; KBDDBUF    RAM
0322 00 1380          DB      00H      ; KEYLOC     RAM
0023 1381 NOVALS EQU      $-INVALS
1382 SIZECHK
0023 1385+ SIZE SET 35
1386+;
1387+; *****
1396 $EJECT

```

LOC	OBJ	LINE	SOURCE STATEMENT	OBJECT STATEMENT	LOC
		1397 \$	INCLUDE(:F0:PARSER.MOD)		
		=1398	CODEBLK 45		
0000		=1403+	ORG 0		
		=1407 ; INIT	INITIALIZES PROCESSOR REGISTERS		
		=1408 ;	AND RAM LOCATIONS TO DEFINED VALUES.		
0000 C5		=1409 INIT:	SEL R00		
0001 BF00		=1410	MOV XPCODE, #0		
0003 74D1		=1411	CALL XPTEST		
0005 27		=1412	CLR A		
0006 3D		=1413	MOVD PSEGLO, A		
0007 3E		=1414	MOVD PSEGH1, A		
0008 B81A		=1415	MOV R0, #1AH ; START AT R01 (REG2) = RAM LOC 1AH		
000A B923		=1416	MOV R1, #LOW NOVALS		
000C BA00		=1417	MOV R2, #LOW INVALS		
000E FA		=1418 INITLP:	MOV A, R2		
000F E3		=1419	MOVP3 A, @R		
0010 A0		=1420	MOV @R0, A		
0011 18		=1421	INC R0		
0012 1A		=1422	INC K2		
0013 E90E		=1423	DJNZ R1, INITLP		
0015 55		=1424	STRT T		
0016 744F		=1425	CALL EPBRK		
0018 B8BB		=1426	MOV R0, #LOW(OV1BAS+OV5SIZE)		
001A 746A		=1427	CALL OVLORD		
001C 54E5		=1428	CALL COMFIL		
001E B937		=1429	MOV R1, #TYPE		
0020 11		=1430	INC @R1		
0021 34F2		=1431	CALL INC5MA		
0023 54E5		=1432	CALL COMFIL		
0025 99EF		=1433	ANL P1, #(NOT EPRSET) ; REMOVE EP RESET SIGNAL.		
0027 0429		=1434	JMP MAIN		
		=1435 ;			
		=1436	SIZECHK		
0029		=1439+ SIZE SET 41			
		=1440+;			
		=1441+; *****			
		=1450 \$EJECT			

LOC	OBJ	LINE	SOURCE STATEMENT	OBJECT STATEMENT	LOC	OBJ
=1451 ;				INCLUDE:PB:PARSER.MOD	0000	
=1452 ;			KEYBOARD LAYOUT:	CODEBK 40	0000	
=1453 ;			=====	0	0000	
=1454 ;				INITIALISE PROCESSOR REGISTERS	0000	
=1455 ;				AND AND:OUTPUS TO RETNED VALUES	0000	
=1456 ;	!	!!	!!	!!	0000	
=1457 ;	!	LIST	!!GO/RESET!!	GO	!!EXAM/CHO!!	0000
=1458 ;	!	!!	!!	!!	!!	0000
=1459 ;						0000
=1460 ;						0000
=1461 ;	!	!!	!!PROG BRK!!	!!PROG MEM!!	!!REGISTER!!	0000
=1462 ;	!	UPLOAD	!!	!!	!!	0000
=1463 ;	!	!!	!!AUTO STP!!	!!SING STP!!	!!NO BRK!!	0000
=1464 ;						0000
=1465 ;						0000
=1466 ;	!	!!	!!DATA BRK!!	!!DATA MEM!!	!!	0000
=1467 ;	!	DNLOAD	!!	!!	!!CLR/PREV!!	0000
=1468 ;	!	!!	!!AUTO BRK!!	!!WITH BRK!!	!!	0000
=1469 ;						0000
=1470 ;						0000
=1471 ;	!	!!	!!	!!	!!	0000
=1472 ;	!	FILL	!!HARD REG!!	NEXT/	!!END/	0000
=1473 ;	!	!!	!!	!!	!!	0000
=1474 ;						0000
=1475 ;						0000
=1476			\$JECT			0000

LOC	OBJ	LINE	SOURCE STATEMENT	THINKSTAT3 300002	LINE	LOC	OBJ
		=1477 ;					
		=1478 ;	THE FOLLOWING EQUATES DETERMINES HOW THE PARSER INTERPRETS				
		=1479 ;	VALUES RETURNED BY THE KEYBOARD SCANNING INPUT ROUTINE				
		=1480 ;	WHEN THE VARIOUS KEYS OF THE KEYBOARD ARE PRESSED				
		=1481 ;					
		=1482 ;					
		=1483 ;	KEY0 EQU 00H VALUE RETURNED FOR EACH KEY OF KEYBOARD MATRIX #				
		=1484 ;	KEY1 EQU 01H BY KEYBOARD SCANNING SUBROUTINE "KBDIN"				
		=1485 ;	KEY2 EQU 02H				
		=1486 ;	KEY3 EQU 03H				
		=1487 ;	KEY4 EQU 04H ! 1C ! 1D ! 1E ! 1F ! ! 0C ! 0D ! 0E ! 0F !				
		=1488 ;	KEY5 EQU 05H				
		=1489 ;	KEY6 EQU 06H ! 18 ! 19 ! 1A ! 1B ! ! 08 ! 09 ! 0A ! 0B !				
		=1490 ;	KEY7 EQU 07H				
		=1491 ;	KEY8 EQU 08H ! 14 ! 15 ! 16 ! 17 ! ! 04 ! 05 ! 06 ! 07 !				
		=1492 ;	KEY9 EQU 09H				
		=1493 ;	KEYA EQU 0AH ! 10 ! 11 ! 12 ! 13 ! ! 00 ! 01 ! 02 ! 03 !				
		=1494 ;	KEYB EQU 0BH				
		=1495 ;	KEYC EQU 0CH				
		=1496 ;	KEYD EQU 0DH				
		=1497 ;	KEYE EQU 0EH				
		=1498 ;	KEYF EQU 0FH				
0010		=1499	KEYFIL EQU 10H ;[FILL COMMAND]				
0012		=1500	KEYNXT EQU 12H ;[NEXT/,]				
0013		=1501	KEYEND EQU 13H ;[END/,]				
0014		=1502	KEYREL EQU 14H ;[DOWNLOAD COMMAND]				
0015		=1503	KEYPAT EQU 15H ;[AUTOBREAK MODIFIER]				
0016		=1504	KEYDM EQU 16H ;[DATA MEMORY MODIFIER]				
0017		=1505	KEYCLR EQU 17H ;[CLEAR/PREVIOUS]				
0018		=1506	KEYREC EQU 18H ;[UPLOAD COMMAND]				
0019		=1507	KEYTRA EQU 19H ;[AUTOSTEP MODIFIER]				
001A		=1508	KEYPM EQU 1AH ;[PROGRAM MEMORY MODIFIER]				
001B		=1509	KEYREG EQU 1BH ;[REGISTER MEMORY MODIFIER]				
001C		=1510	KEYLST EQU 1CH ;[FORMATTED DATA OUTPUT COMMAND]				
001D		=1511	KCORES EQU 1DH ;[GO FROM RESET STATE COMMAND]				
001E		=1512	KEYGO EQU 1EH ;[GO COMMAND]				
001F		=1513	KEYMOD EQU 1FH ;[EXAMINE/MODIFY COMMAND]				
000B		=1514	KSETB EQU 0BH ;[SET BREAKPOINT COMMAND]				
000C		=1515	KCLRB EQU 0CH ;[CLEAR BREAKPOINT COMMAND]				
		=1516 ;					
		=1517 ;					
0019		=1518	PBRK EQU 19H ;[PROGRAM BREAKPOINT MEMORY MODIFIER]				
0015		=1519	DBRK EQU 15H ;[DATA BREAKPOINT MEMORY MODIFIER]				
0011		=1520	RINT EQU 11H ;[HARDWARE REGISTER MEMORY MODIFIER]				
001B		=1521	NOBRK EQU 1BH ;[WITHOUT BREAKPOINTS MODIFIER]				
0016		=1522	WBRK EQU 16H ;[WITH BREAKPOINTS ENABLED MODIFIER]				
001A		=1523	SING EQU 1AH ;[SINGLE STEP MODIFIER]				
		=1524 ;					
		=1525	\$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	ASSEMBLY	LINE	LOC
0059	1C	=1625	INC ITMP	MOV R1, #0	+1271=	0059
005A	FC	=1626	MOV R, ITMP	MOV R, ITMP	+1272=	005A
005B	E3	=1627	MOVFP3 R, 0A ; GET OPTION POINTER	MOVFP3 R, 0A	+1273=	005B
		=1628	MMOV OPTION, A	MMOV OPTION, A	+1274=	005C
005C	B939	=1641+	MOV R1, #OPTION	MOV R1, #OPTION	+1275=	005C
005E	A1	=1642+	MOV @R1, A	MOV @R1, A	+1276=	005E
005F	1C	=1646	INC ITMP	INC ITMP	+1277=	005F
0060	FC	=1647	MOV R, ITMP	MOV R, ITMP	+1278=	0060
0061	E3	=1648	MOVFP3 R, 0A ; GET NO OF PARAMETERS	MOVFP3 R, 0A	+1279=	0061
		=1649	MMOV NUMCON, A	MMOV NUMCON, A	+1280=	0062
0062	B938	=1662+	MOV R1, #NUMCON	MOV R1, #NUMCON	+1281=	0062
0064	A1	=1663+	MOV @R1, A	MOV @R1, A	+1282=	0064
		=1667 ;			+1283=	
		=1668 ;	PARAMETER_BUFFER(0=>5):=0		+1284=	
		=1669 ;			+1285=	
0065	B906	=1670	MOV R1, #6 ; EACH PARAM IS 2 BYTES	MOV R1, #6	+1286=	0065
0067	B030	=1671	MOV R0, #SMALO ; START OF PARAM BUFFERS	MOV R0, #SMALO	+1287=	0067
0069	B000	=1672 MAINB:	MOV @R0, #00H	MOV @R0, #00H	+1288=	0069
006B	18	=1673	INC R0	INC R0	+1289=	006B
006C	E969	=1674	DJNZ K1, MAINB	DJNZ K1, MAINB	+1290=	006C
006E	14EC	=1675	CALL INPKEY	CALL INPKEY	+1291=	006E
		=1676 ;			+1292=	
		=1677 ;	WHILE KEY>MEM(OPTION+TYPE)[6-0] DO		+1293=	
		=1678 ;	IF MEM(OPTION+TYPE)[7]=1 GOTO MAIND1		+1294=	
		=1679 ;	TYPE:=TYPE+1		+1295=	
		=1680 ;	ENDWHILE		+1296=	
		=1681 ;			+1297=	
		=1682	MMOV ITMP, OPTION	MMOV ITMP, OPTION	+1298=	
0070	B939	=1690+	MOV R1, #OPTION	MOV R1, #OPTION	+1299=	0070
0072	F1	=1699+	MOV R, @R1	MOV R, @R1	+1300=	0072
0073	NC	=1712+	MOV ITMP, A	MOV ITMP, A	+1301=	0073
0074	1C	=1715	INC ITMP	INC ITMP	+1302=	0074
		=1716 MAINC1:	MOV R, ITMP	MOV R, ITMP	+1303=	0075
0075	FC	=1732+	MOV R, ITMP	MOV R, ITMP	+1304=	0075
0076	E3	=1736	MOVFP3 R, 0A	MOVFP3 R, 0A	+1305=	0076
0077	97	=1737	CLR C	CLR C	+1306=	0077
0078	F7	=1738	RLC A	RLC A	+1307=	0078
0079	77	=1739	RR A ; STRIP BIT SEVEN INTO CARRY	RR A	+1308=	0079
007A	D8	=1740	XRL R, KEY	XRL R, KEY	+1309=	007A
007B	C693	=1741	JZ MAIND	JZ MAIND	+1310=	007B
007D	F687	=1742	JC MAIND1	JC MAIND1	+1311=	007D
		=1743	MINC TYPE	MINC TYPE	+1312=	007E
007F	B937	=1748+	MOV K1, #TYPE	MOV K1, #TYPE	+1313=	007F
0081	F1	=1749+	MOV R, @R1	MOV R, @R1	+1314=	0081
0082	17	=1753+	INC A	INC A	+1315=	0082
0083	A1	=1758+	MOV @R1, A	MOV @R1, A	+1316=	0083
0084	1C	=1761	INC ITMP	INC ITMP	+1317=	0084
0085	0475	=1762	JMP MAINC1	JMP MAINC1	+1318=	0085
		=1763 ;			+1319=	
		=1764 ;	MODIFIER NOT FOUND SO RESET TYPE INDEX TO DEFAULT CASE (ZERO).		+1320=	
		=1765 ;			+1321=	
		=1766 MAIND1:	MMOV TYPE, ZERO	MMOV TYPE, ZERO	+1322=	
0087	B937	=1777+	MOV R1, #TYPE	MOV R1, #TYPE	+1323=	0087
0089	B100	=1778+	MOV @R1, #ZERO	MOV @R1, #ZERO	+1324=	0089
		=1782	MMOV R, OPTION	MMOV R, OPTION	+1325=	

LOC	OBJ	LINE	SOURCE STATEMENT	LOC	OBJ
008B	B939	=1791+	MOV R1, #OPTION	008B	B939
008D	F1	=1792+	MOV A, @R1	008D	F1
008E	E3	=1796	MOVFP3 A, @R1	008E	E3
008F	3404	=1797	CALL OUTMSG	008F	3404
0091	049E	=1798	JMP MAINB0	0091	049E
		=1799 ;			
		=1800 ;	CALL OUTPUT_MESSAGE(MODIFIER)		
		=1801	MAIND: MMOV A, OPTION		
0093	B939	=1810+	MOV R1, #OPTION	0093	B939
0095	F1	=1811+	MOV A, @R1	0095	F1
0096	E3	=1815	MOVFP3 A, @R1	0096	E3
		=1816	MADD A, TYPE		
0097	B937	=1822+	MOV R1, #TYPE	0097	B937
0099	61	=1823+	ADD A, @R1	0099	61
009A	3404	=1827	CALL OUTMSG	009A	3404
009C	14EC	=1828	CALL INPKEY	009C	14EC
		=1829 ;			
009E	BC00	=1830	MAINB0: MOV ITMP, #0	009E	BC00
00A0	2330	=1831	MAINB1: MOV A, #SMAL0	00A0	2330
00A2	6C	=1832	ADD A, ITMP	00A2	6C
00A3	6C	=1833	ADD A, ITMP	00A3	6C
00A4	A8	=1834	MOV R0, A	00A4	A8
00A5	14C0	=1835	CALL INPADR	00A5	14C0
00A7	FGBA	=1836	JC CHDINT	00A7	FGBA
00A9	1C	=1837	INC ITMP	00A9	1C
00AA	B938	=1838	MOV R1, #NUMCON	00AA	B938
00AC	F1	=1839	MOV A, @R1	00AC	F1
00AD	07	=1840	DEC A	00AD	07
00AE	A1	=1841	MOV @R1, A	00AE	A1
00AF	C6BA	=1842	JZ CHDINT	00AF	C6BA
00B1	FB	=1843	MOV A, KEY	00B1	FB
00B2	D313	=1844	XRL A, #KEYEND	00B2	D313
00B4	C6BA	=1845	JZ CHDINT	00B4	C6BA
00B6	14EC	=1846	CALL INPKEY	00B6	14EC
00B8	04A0	=1847	JMP MAINB1	00B8	04A0
		=1848 ;			
		=1849 ;	CHDINT ENTER THE COMMAND PROCESSOR WITH:		
		=1850 ;	BASE_CODE=THE MAIN COMMAND TYPE		
		=1851 ;	TYPE=SUBCOMMAND TYPE		
		=1852 ;	PARAMETER(1)=FIRST ADDRESS		
		=1853 ;	PARAMETER(2)=SECOND ADDRESS		
		=1854 ;	PARAMETER(3)=DATA		
00BA	4400	=1855	CHDINT: JMP IMPLM	00BA	4400
		=1856 ;			
		=1857 ;	MERRR ERROR ENCOUNTERED IN MAIN PARSING ROUTINE		
00BC	BA01	=1858	MERRR: MOV LDATA, #1	00BC	BA01
00BE	249A	=1859	JMP PERRR	00BE	249A
		=1860	SIZECHK		
0097		=1863+	SIZE SET 151	0097	
		=1864+;			
		=1865+;	*****		
		=1874	\$EJECT		

LOC	OBJ	LINE	SOURCE STATEMENT	LINE	LOC
		=1875	DATABLK 50		
0323		=1880+	ORG 803		
		=1884 ;			
		=1885 ; *****			
		=1886 ;			
		=1887 ;	TABLES FOR PARSER		
		=1888 ;			
		=1889 ; *****			
		=1890 ;			
		=1891 ;	THE CTAB TABLE CONTAINS <COMSIZ> ENTRIES FOR EACH COMMAND. THE MEANING		
		=1892 ;	OF THE ENTRIES IS AS FOLLOWS:		
		=1893 ;			
		=1894 ;	ENTRY 0. COMMAND KEY TO INITIATE		
		=1895 ;	ENTRY 1. POINTER TO THE LIST OF OPTIONS APPLICABLE TO THIS COMMAND		
		=1896 ;	ENTRY 2. NUMBER OF NUMERIC PARAMETERS REQUIRED BY THE COMMAND		
		=1897 ;			
0023		=1898 CTAB	EQU \$ AND OFFH		
0003		=1899 COMSIZ	EQU 3		
		=1900 ;			
0323 1F		=1901	DB KEYMOD, LOW OPTAB1, 1 ; EXAM		
0324 3F		=			
0325 01		=			
0326 1E		=1902	DB KEYGO, LOW OPTAB3, 1 ; GO		
0327 49		=			
0328 01		=			
0329 10		=1903	DB KEYFIL, LOW OPTAB1, 3 ; FILL		
032A 3F		=			
032B 03		=			
032C 1C		=1904	DB KEYLST, LOW OPTAB1, 2 ; DUMP		
032D 3F		=			
032E 02		=			
032F 18		=1905	DB KEYREC, LOW OPTAB1, 2 ; RECORD		
0330 3F		=			
0331 02		=			
0332 14		=1906	DB KEYREL, LOW OPTAB1, 0 ; RELOAD		
0333 3F		=			
0334 00		=			
0335 00		=1907	DB KSE1B, LOW OPTAB2, 1 ; SETBRK		
0336 46		=			
0337 01		=			
0338 0C		=1908	DB KCLRB, LOW OPTAB2, 1 ; CLRBRK		
0339 46		=			
033A 01		=			
033B 1D		=1909	DB KGORES, LOW OPTAB3, 0 ; GO FROM RESET STATE		
033C 49		=			
033D 00		=			
033E FF		=1910	DB OFFH ; ESCOP		
		=1911 ;			
		=1912 \$EJECT			


```

=1916 ; *****
=1917 ; ENTRY 0. START OF TABLE OF MODIFIER RESPONSES.
=1918 ; ENTRY 1+. ALLOWED MODIFIER KEYSTROKES CORRESPONDING TO OPTIONS 0-5.
=1919 ; NOTE THAT THE LAST BYTE IN EACH OPTION GROUP HAS BIT
=1920 ; SEVEN SET TO INDICATE THE END.
=1921 ; *****
033F 26 =1922 OPTAB1: DB STRMEM
0340 1A =1923 DB KEYPM, KEYDM, KEYREG, RINT
0341 16 =
0342 1B =
0343 11 =
0344 19 =1924 DB PBK, DBK OR 80H
0345 95 =
0346 26 =1925 OPTAB2: DB STRMEM
0347 1A =1926 DB KEYPM, KEYDM OR 80H
0348 96 =
0349 2C =1927 OPTAB3: DB STRGOC
034A 1B =1928 DB NOBRK, WBRK, SING
034B 16 =
034C 1A =
034D 15 =1929 DB KEYPAT, KEYTRA OR 80H
034E 99 =
002C =1930 SIZECHK
=1933+ SIZE SET 44
=1934+;
=1935+; *****
=1944 $EJECT

```

LOC	OBJ	LINE	SOURCE STATEMENT	OBJECT CODE	LINE	LOC
		=1945	CODEBLK 130			
0100		=1955+	ORG 256			
		=1959	OUTUTL OUTPUT ONE OF FOUR UTILITY DISPLAY PROMPTS (LEFT JUSTIFIED)			
		=1960 ;	ACCORDING TO ACC CONTENTS (0-3).			
		=1961 ; OUTCLR	CLEAR DISPLAY AND OUTPUT CHARACTER STRING STARTING			
		=1962 ;	AT THE ADDRESS POINTED TO BY BYTE AT ADDRESS IN ACCUMULATOR.			
		=1963 ; OUTMSG	SUBROUTINE TO COPY A STRING OF BIT PATTERNS FROM ROM TO THE			
		=1964 ;	DISPLAY REGISTERS.			
		=1965 ;	STRING SELECTED IS DETERMINED BY ACC WHEN CALLED.			
		=1966 ;	ON ENTERING OUTMSG, ACC CONTENTS ARE USED TO ADDRESS A BYTE IN A			
		=1967 ;	LOOKUP TABLE ON THE CURRENT PAGE WHICH CONTAINS THE ADDRESS OF			
		=1968 ;	A STRING OF SEGMENT PATTERN DATA BYTES TO BE PRINTED ONTO THE			
		=1969 ;	DISPLAY.			
		=1970 ;	THE END OF THE STRING IS INDICATED WHEN BIT7 =1			
		=1971 ;	CALLS SUBROUTINE 'WDISP'			
		=1972 ;	TO ACTUALLY EFFECT WRITING INTO THE DISPLAY REGISTERS.			
0100 0319		=1973 OUTUTL:	ADD A, #STRUTL			
0102 04F1		=1974 OUTCLR:	CALL CLEAR			
0104 A3		=1975 OUTMSG:	MOVP A, 0A			
		=1976	MMOV STRTMP, A			
0105 B940		=1989+	MOV R1, #STRTMP			
0107 A1		=1990+	MOV @R1, A			
		=1994 PRNT2:	MMOV A, STRTMP ; LOAD NEXT CHARACTER LOCATION			
0108 B940		=2003+	MOV R1, #STRTMP			
010A F1		=2004+	MOV A, @R1			
010B A3		=2008	MOVP A, 0A ; LOAD BIT PATTERN INDIRECT			
010C F217		=2009	JB7 PRNT1			
010E D4D8		=2010	CALL WDISP ; OUTPUT TO NEXT CHARACTER POSITION			
		=2011	MINC STRTMP ; INDEX POINTER			
0110 B940		=2016+	MOV R1, #STRTMP			
0112 F1		=2017+	MOV A, @R1			
0113 17		=2021+	INC A			
0114 A1		=2026+	MOV @R1, A			
0115 2408		=2029	JMP PRNT2			
0117 C4D8		=2030 PRNT1:	JMP WDISP ; DONE			
		=2031 ;				
0019		=2032 STRUTL EQU	LOW \$			
0119 31		=2033	DB LOW(DERROR) ; UTILITY MESSAGE 0 ADDRESS			
011A 37		=2034	DB LOW(DSGNON) ; UTILITY MESSAGE 1 ADDRESS			
011B 3E		=2035	DB LOW(DRUN) ; UTILITY MESSAGE 2 ADDRESS			
011C 44		=2036	DB LOW(DBPNT) ; UTILITY MESSAGE 3 ADDRESS			
001D		=2037 STRCOM EQU	LOW \$			
011D 46		=2038	DB LOW(CMOD) ; BASIC COMMAND 0 RESPONSE ADDRESS			
011E 49		=2039	DB LOW(DGO) ; BASIC COMMAND 1 RESPONSE ADDRESS			
011F 4B		=2040	DB LOW(DFILL) ; BASIC COMMAND 2 RESPONSE ADDRESS			
0120 4E		=2041	DB LOW(DLST) ; BASIC COMMAND 3 RESPONSE ADDRESS			
0121 51		=2042	DB LOW(DREC) ; BASIC COMMAND 4 RESPONSE ADDRESS			
0122 54		=2043	DB LOW(DREL) ; BASIC COMMAND 5 RESPONSE ADDRESS			
0123 57		=2044	DB LOW(DSB) ; BASIC COMMAND 6 RESPONSE ADDRESS			
0124 5A		=2045	DB LOW(DCB) ; BASIC COMMAND 7 RESPONSE ADDRESS			
0125 5D		=2046	DB LOW(DGR) ; BASIC COMMAND 8 RESPONSE ADDRESS			
0026		=2047 STRMEM EQU	LOW \$			
0126 5F		=2048	DB LOW(DPRMEM) ; DATA TYPE MODIFIER 0 RESPONSE ADDRESS			
0127 61		=2049	DB LOW(DDMEM) ; DATA TYPE MODIFIER 1 RESPONSE ADDRESS			
0128 63		=2050	DB LOW(DRM) ; DATA TYPE MODIFIER 2 RESPONSE ADDRESS			

LOC	OBJ	LINE	SOURCE STATEMENT	LINE	LOC
0129 69		=2051	DB LOW(DINTRG) ; DATA TYPE MODIFIER 3 RESPONSE ADDRESS		
012A 65		=2052	DB LOW(DPRBRK) ; DATA TYPE MODIFIER 4 RESPONSE ADDRESS		
012B 67		=2053	DB LOW(DDABRK) ; DATA TYPE MODIFIER 5 RESPONSE ADDRESS		
002C		=2054	STRGOC EQU LOW \$		
012C 6B		=2055	DB LOW(DNOBRK) ; EXECUTION MODE MODIFIER 0		
012D 6D		=2056	DB LOW(DWBRK) ; EXECUTION MODE MODIFIER 1		
012E 6F		=2057	DB LOW(DSS) ; EXECUTION MODE MODIFIER 2		
012F 72		=2058	DB LOW(DPA) ; EXECUTION MODE MODIFIER 3		
0130 75		=2059	DB LOW(DTR) ; EXECUTION MODE MODIFIER 4		
		=2060 ;			
		=2061 ;	UTILITY OUTPUT MESSAGES		
		=2062 ;			
		=2063	ERROR:		
0131 79		=2064	DB 01111001B ; "E"		
0132 50		=2065	DB 01010000B ; "R"		
0133 50		=2066	DB 01010000B ; "R"		
0134 5C		=2067	DB 01011100B ; "O"		
0135 50		=2068	DB 01010000B ; "R"		
0136 C0		=2069	DB 11000000B ; "-."		
		=2070	DSGNON:		
0137 00		=2071	DB 00000000B ; " "		
0138 76		=2072	DB 01110110B ; "H"		
0139 6D		=2073	DB 01101101B ; "S"		
013A 79		=2074	DB 01111001B ; "E"		
013B 40		=2075	DB 01000000B ; "-."		
013C 66		=2076	DB 01100110B ; "4"		
013D E7		=2077	DB 11100111B ; "9."(TH)		
		=2078	DRUN:		
013E 00		=2079	DB 00000000B ; " "		
013F 40		=2080	DB 01000000B ; "-."		
0140 50		=2081	DB 01010000B ; "R"		
0141 1C		=2082	DB 00011100B ; "U"		
0142 54		=2083	DB 01010100B ; "N"		
0143 C0		=2084	DB 11000000B ; "-."		
		=2085	DEPNT:		
0144 73		=2086	DB 01110011B ; "P"		
0145 B9		=2087	DB 10111001B ; "C."		
		=2088	\$EJECT		

LOC	OBJ	LINE	SOURCE STATEMENT	THIRTYTWO	THIRTY	THIRTY
		=2089 ;				
		=2090 ;	PRIMARY COMMAND RESPONSE STRING PATTERNS			
		=2091 ;				
		=2092 DMOD:				
0146 79		=2093	DB 001111001B, 001111001B, 111101000B ; "ECH."			
0147 39		=				
0148 F4		=				
		=2094 DGO:				
0149 3D		=2095	DB 00111101B, 110111000B ; "GO."			
014A DC		=				
		=2096 DFILL:				
014B 71		=2097	DB 01110001B, 00110000B, 101110000B ; "FIL."			
014C 30		=				
014D B8		=				
		=2098 DLST:				
014E 38		=2099	DB 00111000B, 01101101B, 11111000B ; "LST."			
014F 6D		=				
0150 F8		=				
		=2100 DREC:				
0151 3E		=2101	DB 00111110B, 01110011B, 10111000B ; "UPL."			
0152 73		=				
0153 B8		=				
		=2102 DREL:				
0154 5E		=2103	DB 01011110B, 01010100B, 10111000B ; "DNL."			
0155 54		=				
0156 B8		=				
		=2104 DSB:				
0157 6D		=2105	DB 01101101B, 01111000B, 11111100B ; "STB."			
0158 78		=				
0159 FC		=				
		=2106 DCD:				
015A 39		=2107	DB 00111001B, 00111000B, 11111100B ; "CLB."			
015B 38		=				
015C FC		=				
		=2108 DGR:				
015D 3D		=2109	DB 00111101B, 11010000B ; "GR."			
015E D0		=				
		=2110 \$EJECT				

```

-2113 ;
=2114 DFRMEM:
015F 73 =2115 DB 01110011B, 11010000B ; "PR."
0160 D0 =
=2116 DDANEM:
0161 5E =2117 DB 01011110B, 11110111B ; "DA."
0162 F7 =
=2118 DRM:
0163 50 =2119 DB 01010000B, 10111101B ; "RG."
0164 B0 =
=2120 DFRBK:
0165 73 =2121 DB 01110011B, 11111100B ; "PB."
0166 FC =
=2122 DDABRK:
0167 5E =2123 DB 01011110B, 11111100B ; "DE."
0168 FC =
=2124 DINTRG:
0169 76 =2125 DB 01110110B, 11010000B ; "HR."
016A D0 =
=2126 ;
=2127 ; RESPONSE MESSAGES FOR GO CONDITION MODIFIERS.
=2128 ;
=2129 DNOBRK:
016B 54 =2130 DB 01010100B, 11111100B ; "NB."
016C FC =
=2131 DWRBK:
016D 7C =2132 DB 01111100B, 11010000B ; "BR."
016E D0 =
=2133 DSS:
016F 6D =2134 DB 01101101B, 01101101B, 11111000B ; "SST."
0170 6D =
0171 F8 =
=2135 DPA:
0172 77 =2136 DB 01110111B, 01111100B, 11010000B ; "ABR."
0173 7C =
0174 D0 =
=2137 DTR:
0175 77 =2138 DB 01110111B, 01101101B, 11111000B ; "AST."
0176 6D =
0177 F8 =
=2139 ;
=2140 SIZECHK
0078 =2143+ SIZE SET 120
=2144+;
=2145+; *****
=2154 $EJECT

```


LOC	OBJ	LINE	SOURCE STATEMENT	OBJECT STATEMENT	LOC	OBJ
		=2155	CODECLK 45	CODECLK 45		
00C0		=2160+	ORG 192	ORG 192		
		=2164 ;	INPADR INPUT DATA INTO TWO-BYTE PARAMETER BUFFER INDICATED BY R0.			
		=2165 ;	RECEIVE NUMERIC KEYS FROM KEYBOARD UNTIL ' ' OR ' . '.			
		=2166 ;	SHIFT INTO ADDRESS BUFFER;			
		=2167 ;	RE-WRITE DISPLAY.			
		=2168 ;	IF NUMBER OF CONSTANTS NEEDED IS ZERO, NO NEW PARAMETERS ARE ALLOWED.			
		=2169 ;				
00C0 97		=2170	INPADR: CLR C	INPADR: CLR C		
00C1 A7		=2171	CPL C	CPL C		
		=2172	MMOV A, NUMCON	MMOV A, NUMCON		
00C2 B938		=2181+	MOV R1, #NUMCON	MOV R1, #NUMCON		
00C4 F1		=2182+	MOV A, @R1	MOV A, @R1		
00C5 C6D7		=2186	JZ ELSIF1	JZ ELSIF1		
00C7 FB		=2187	INPAD1: MOV A, KEY	INPAD1: MOV A, KEY		
00C8 92D7		=2188	JB4 ELSIF1	JB4 ELSIF1		
00CA 20		=2189	XCH A, @R0	XCH A, @R0		
00CB 47		=2190	SWAP A	SWAP A		
00CC 20		=2191	XCH A, @R0	XCH A, @R0		
00CD 30		=2192	XCHD A, @R0	XCHD A, @R0		
00CE 18		=2193	INC R0	INC R0		
00CF 30		=2194	XCHD A, @R0	XCHD A, @R0		
00D0 3478		=2195	CALL UPDADR	CALL UPDADR		
00D2 14EC		=2196	CALL INPKEY	CALL INPKEY		
00D4 97		=2197	CLR C	CLR C		
00D5 04C7		=2198	JMP INPAD1	JMP INPAD1		
		=2199 ;				
		=2200 ;	ELSIF1 IF KEY=' ' OR ' . ' THEN RETURN.			
		=2201 ;				
00D7 FB		=2202	ELSIF1: MOV A, KEY	ELSIF1: MOV A, KEY		
00D8 D312		=2203	XRL A, #KEYNXT	XRL A, #KEYNXT		
00DA C6E5		=2204	JZ ELSIF2	JZ ELSIF2		
00DC FB		=2205	MOV A, KEY	MOV A, KEY		
00DD D313		=2206	XRL A, #KEYEND	XRL A, #KEYEND		
00DF C6E5		=2207	JZ ELSIF2	JZ ELSIF2		
		=2208 ;				
		=2209 ;	ELSE GOTO PERROR.			
		=2210 ;				
00E1 B802		=2211	MOV LDATA, #2	MOV LDATA, #2		
00E3 249A		=2212	JMP PERROR	JMP PERROR		
00E5 B846		=2213	ELSIF2: MOV R0, #SEGMAP	ELSIF2: MOV R0, #SEGMAP		
00E7 B903		=2214	MOV R1, #3	MOV R1, #3		
00E9 B4F5		=2215	CALL DBLANK	CALL DBLANK		
00EB 83		=2216	RET	RET		
		=2217	SIZECHK			
002C		=2220+	SIZE SET 44			
		=2221+;				
		=2222+;	*****			
		=2231	\$EJECT			

LOC	OBJ	LINE	SOURCE STATEMENT	OBJECT STATEMENT	LINE	LOC	OBJ
		=2232	CODEBLK 35	CODEBLK 35	=2232		
0178		=2242+	ORG 376	ORG 376	=2242+		
		=2246	UPDADR: UPDATE ADDRESS FIELD	UPDADR: UPDATE ADDRESS FIELD	=2246		
		=2247	(LAST THREE CHARACTERS OF DISPLAY) WITH ADDRESS BUFFER	(LAST THREE CHARACTERS OF DISPLAY) WITH ADDRESS BUFFER	=2247		
		=2248	UPDADR: MOV NEXTPL, PLUS3	UPDADR: MOV NEXTPL, PLUS3	=2248		
0178 B93A		=2259+	MOV R1, #NEXTPL	MOV R1, #NEXTPL	=2259+		
017A D103		=2260+	MOV @R1, #PLUS3	MOV @R1, #PLUS3	=2260+		
		=2264	WRITE ADDR INTO NEXT THREE BUFFER LOCATIONS.	WRITE ADDR INTO NEXT THREE BUFFER LOCATIONS.	=2264		
017C F0		=2265	UPDADR: MOV A, @R0	UPDADR: MOV A, @R0	=2265		
017D C8		=2266	DEC R0	DEC R0	=2266		
017E 530F		=2267	ANL A, #0FH	ANL A, #0FH	=2267		
0180 968E		=2268	JNZ DSPHI	JNZ DSPHI	=2268		
0182 D4D8		=2269	CALL MDISP	CALL MDISP	=2269		
0184 F0		=2270	MOV A, @R0	MOV A, @R0	=2270		
0185 47		=2271	SWAP A	SWAP A	=2271		
0186 530F		=2272	ANL A, #0FH	ANL A, #0FH	=2272		
0188 9692		=2273	JNZ DSPM1	JNZ DSPM1	=2273		
018A D4D8		=2274	CALL MDISP	CALL MDISP	=2274		
018C 2494		=2275	JMP DSPL0	JMP DSPL0	=2275		
018E D4D3		=2276	DSPHI: CALL DSPRCC	DSPHI: CALL DSPRCC	=2276		
0190 F0		=2277	DSPMID: MOV A, @R0	DSPMID: MOV A, @R0	=2277		
0191 47		=2278	SWAP A	SWAP A	=2278		
0192 D4D3		=2279	DSPM1: CALL DSPRCC	DSPM1: CALL DSPRCC	=2279		
0194 F0		=2280	DSPL0: MOV A, @R0	DSPL0: MOV A, @R0	=2280		
0195 D4D3		=2281	CALL DSPRCC	CALL DSPRCC	=2281		
0197 83		=2282	RET	RET	=2282		
		=2283	SIZECHK	SIZECHK	=2283		
0020		=2286+	SIZE SET 32	SIZE SET 32	=2286+		
		=2287+			=2287+		
		=2288+	*****	*****	=2288+		
		=2297	\$EJECT	\$EJECT	=2297		

		REPERI			
	=2312 ;	ERROR:	OUTPUT_MESSAGE(PERROR_PROMPT)		
	=2313 ;		OUTPUT(LDATA)		
	=2314 ;		CALL INPUT_BYTE(KEY)		
	=2315 ;		UNTIL KEY='CLEAR/PKEYIOUS'		
0190 0A04	=2317	ERROR:	MOV LDATA, #4		
019A BF02	=2318	ERROR:	MOV XPCODE, #2		
019C 74D1	=2319		CALL XPTST		
019E 27	=2320		CLR A		
019F D7	=2321		MOV PSW, A		
01A0 FB	=2322		MOV A, KEY		
01A1 D317	=2323		XRL A, #KEYCLR		
01A3 C606	=2324		JZ ERROR2		
01A5 27	=2325		CLR A		
01A6 3400	=2326		CALL OUTUTL		
01A8 FA	=2327		MOV A, LDATA		
01A9 D4D3	=2328		CALL DSPACC		
	=2329		MMOV KBD0UF, NEG1		
01AB 093B	=2340+		MOV R1, #KBD0UF		
01AD B1FF	=2341+		MOV R1, #NEG1		
01AF 14EC	=2345		CALL INPKEY		
01B1 FB	=2346		MOV A, KEY		
01B2 D313	=2347		XRL A, #KEYEND		
01B4 9698	=2348		JNZ RRROR		
01B6 0429	=2349	ERROR2:	JMP MAIN		
	=2350		SIZECHK		
0020	=2353+	SIZE	SET 32		
	=2354+				
	=2355+		*****		
	=2364 ;				
	=2365		CODEBLK 00		
0200	=2380+	ORG	512		
	=2384 ;	IMPLEM	IMPLEMENT COMMAND		
0200 2306	=2385	IMPLEM:	MOV A, #LOW(JMPTBL)		
	=2386		MADD A, BCODE		
0202 0936	=2392+		MOV R1, #BCODE		
0204 61	=2393+		ADD A, R1		
0205 B3	=2397		JMPP 0A		
	=2398 ;				
	=2399	JMPTBL:			
0206 0F	=2400	DB	LOW(JTOMOD)		
0207 20	=2401	DB	LOW(JTGOO)		
0208 22	=2402	DB	LOW(JTJFIL)		
0209 1A	=2403	DB	LOW(JTJLST)		
020A 11	=2404	DB	LOW(JTJREC)		
020B 16	=2405	DB	LOW(JTJREL)		
020C 2C	=2406	DB	LOW(COMSBR)		
020D 28	=2407	DB	LOW(COMCBR)		
020E 26	=2408	DB	LOW(JGORES)		
	=2409 ;				
020F 444F	=2410	JTOMOD:	JMP EXAMIN		
	=2411 ;				
0211 85	=2412	JTJREC:	CLR F0 ; F0=0 ==> HEX FORMAT DATA DUMP		

LOC	OBJ	LINE	SOURCE STATEMENT	DISASSEMBLY	HEX	DEC
		=2516	CODEBLK 75			
024F		=2531	ORG 591			
		=2535	EXAMIN EXAMINE/MODIFY MEMORY COMMAND.			
		=2536	DISPLAYS MEMORY ADDRESS SPACE OPTION, ADDRESS VALUE, AND CURRENT DATA.			
		=2537	READS KEYBOARD AND INTERPRETS RESPONSE.			
		=2538				
		=2539	OUTPUT_MESSAGE(<MEMORY_SPACE_OPTION><SMAR>=<DATA_BYTE>)			
024F 85		=2540	EXAMIN: CLR F0			
		=2541	EXAM0: MMOV A, TYPE			
0250 0937		=2550+	MOV R1, #TYPE			
0252 F1		=2551+	MOV A, @R1			
0253 0326		=2555	ADD A, #STRMEM ; OFFSET FOR FIRST MEMORY TYPE STRING			
0255 3402		=2556	CALL OUTCLR			
0257 0831		=2557	MOV R0, #SMALO+1			
0259 347C		=2558	CALL UPDAD1			
025B 2348		=2559	MOV A, #01001000B ; '='			
025D D4D8		=2560	CALL WDISP			
025F 14FC		=2561	CALL LFETCH			
0261 FA		=2562	MOV A, LDAT4			
0262 47		=2563	SWAP A			
0263 D4D3		=2564	CALL DSPACC			
0265 FA		=2565	MOV A, LDAT4			
0266 D4D3		=2566	CALL DSPACC			
		=2567				
		=2568				
		=2569	INPUT_KEY(KEY)			
		=2570	IF (KEY IS NOT NUMERIC)			
		=2571	IF (KEY=KEYEND) GO TO PARSER			
		=2572	ELSEIF (KEY=KEYNEXT)			
		=2573	INCREMENT <SMAR>			
		=2574	GOTO EXAMIN			
		=2575	ELSEIF (KEY=KEYPREVIOUS)			
		=2576	DECREMENT <SMAR>			
		=2577	GOTO EXAMIN			
		=2578	ELSE GOTO ERROR			
		=2579				
0260 14EC		=2580	CALL INPKEY			
		=2581	MMOV A, KEY			
026A FB		=2597+	MOV A, KEY			
026B 927B		=2601	JB4 EXAM1			
		=2602				
		=2603	APPEND DATA WITH <LOWNIB-<KEY>>			
		=2604	CALL LSTORE			
		=2605	GOTO EXAMIN			
		=2606				
026D FA		=2607	MOV A, LDAT4			
026E 47		=2608	SWAP A			
026F 53F0		=2609	ANL A, #0F0H			
0271 B675		=2610	JF0 EXAM5			
0273 27		=2611	CLR A			
0274 95		=2612	CPL F0			
0275 6B		=2613	EXAM5: ADD A, KEY			
0276 FA		=2614	MOV LDAT4, A			
0277 F400		=2615	CALL LSTORE			
0279 4450		=2616	JMP EXAM0			

LOC	OBJ	LINE	SOURCE STATEMENT	THIRDTY2 33W42	341	LOC 001
		=2617 ;		CODEBLK 75	=2617	
027B	D313	=2618 EXAM1: XRL A, #(KEYEND)		ORG 301	=2618	027B
027D	9681	=2619 JNZ EXAM2		ORG 302	=2619	
027F	0429	=2620 JMP MAIN		ORG 303	=2620	
		=2621 ;		ORG 304	=2621	
0281	FB	=2622 EXAM2: MOV A, KEY		ORG 305	=2622	
0282	D312	=2623 XRL A, #KEYNXT		ORG 306	=2623	
0284	968A	=2624 JNZ EXAM3		ORG 307	=2624	
0286	34F2	=2625 CALL INCSMA		ORG 308	=2625	
0288	444F	=2626 JMP EXAMIN		ORG 309	=2626	
028A	FB	=2627 EXAM3: MOV A, KEY		ORG 310	=2627	
028B	D317	=2628 XRL A, #KEYCLR		ORG 311	=2628	
028D	9693	=2629 JNZ EXAM4		ORG 312	=2629	
028F	54F4	=2630 CALL DECSMA		ORG 313	=2630	
0291	444F	=2631 JMP EXAMIN		ORG 314	=2631	
0293	8A03	=2632 EXAM4: MOV LDATA, #03H		ORG 315	=2632	
0295	249A	=2633 JMP PERROR		ORG 316	=2633	
		=2634 SIZECHK		ORG 317	=2634	
0048		=2637+ SIZE SET 72		ORG 318	=2637	
		=2638+;		ORG 319	=2638	
		=2639+; *****		ORG 320	=2639	
		=2648 ;		ORG 321	=2648	
		=2649 CODEBLK 4		ORG 322	=2649	
00EC		=2654+ ORG 236		ORG 323	=2654	
00EC	D4C2	=2658 INPKEY: CALL KBDIN ; RETURNS KEY DEPRESSION IN A		ORG 324	=2658	
00EE	AD	=2659 MOV KEY, A		ORG 325	=2659	
00EF	83	=2660 RET		ORG 326	=2660	
		=2661 SIZECHK		ORG 327	=2661	
0004		=2664+ SIZE SET 4		ORG 328	=2664	
		=2665+;		ORG 329	=2665	
		=2666+; *****		ORG 330	=2666	
		=2675 \$EJECT		ORG 331	=2675	

LOC	OBJ	LINE	SOURCE STATEMENT	THIRTEENTH SOURCE	LINE	LOC	OBJ
		2676 \$	INCLUDE(:F0:GOCOMS.MOD)				
		=2677	CODEBLK 210				
0400		=2697+	ORG 1024				
		=2701 ;	EPRUN RUN EMULATION MODE.				
		=2702 ;	RELOAD EP WITH SYSTEM STATUS AND RELEASE				
		=2703 ;	SEQUENCE IS AS FOLLOWS:				
		=2704 ;	IF COMMAND WAS TERMINATED BY THE 'NEXT' KEY:				
		=2705 ;	STORE SMA INTO EP PC;				
		=2706 ;	STORE EP PC INTO TOP-OF-STACK (RELATIVE TO EP PSW);				
		=2707 ;	PASS EP R0;				
		=2708 ;	PASS EP PSW;				
		=2709 ;	PASS EP TIMER;				
		=2710 ;	PASS EP ACCUMULATOR;				
		=2711 ;					
0400	2302	=2712 EPRUN:	MOV A, #2				
0402	3400	=2713	CALL OUTUTL				
		=2714	MMOV A, NUMCON				
0404	B938	=2723+	MOV R1, #NUMCON				
0406	F1	=2724+	MOV A, @R1				
0407	9615	=2728	JNZ EPCONT				
		=2729	MMOV EPPCLO, SMALO				
0409	B930	=2745+	MOV R1, #SMALO				
040B	F1	=2746+	MOV A, @R1				
040C	B924	=2752+	MOV R1, #EPPCLO				
040E	A1	=2753+	MOV @R1, A				
		=2756	MMOV EPPCHI, SMAHI				
040F	B931	=2772+	MOV R1, #SMAHI				
0411	F1	=2773+	MOV A, @R1				
0412	B925	=2779+	MOV R1, #EPPCHI				
0414	A1	=2780+	MOV @R1, A				
0415	FB	=2783 EPCONT:	MOV A, KEY				
0416	D312	=2784	XRL A, #KEYNXT				
0418	C61F	=2785	JZ EPCONT				
041A	2301	=2786	MOV A, #01H ; STACK ONE LEVEL DEEP TO HOLD USER STARTING ADDRESS				
		=2787	MMOV EPPSW, A				
041C	B921	=2800+	MOV R1, #EPPSW				
041E	A1	=2801+	MOV @R1, A				
		=2805 EPCONT1:	MMOV LDATA, EPPCLO				
041F	B924	=2821+	MOV R1, #EPPCLO				
0421	F1	=2822+	MOV A, @R1				
0422	AA	=2835+	MOV LDATA, A				
		=2838	MMOV A, EPPSW				
0423	B921	=2847+	MOV R1, #EPPSW				
0425	F1	=2848+	MOV A, @R1				
0426	07	=2852	DEC A				
0427	5307	=2853	ANL A, #07H				
0429	E7	=2854	RL A				
042A	0308	=2855	ADD A, #08H				
		=2856	MMOV SMALO, A				
042C	B930	=2869+	MOV R1, #SMALO				
042E	A1	=2870+	MOV @R1, A				
042F	F4C3	=2874	CALL EPSTOR				
		=2875	MINC SMALO				
0431	B930	=2880+	MOV R1, #SMALO				
0433	F1	=2881+	MOV A, @R1				

0430 H1	=2890+	MOV	R1, A				
	=2893	MMOV	A, EPPSW				
0436 B921	=2902+	MOV	R1, #EPPSW				
0438 F1	=2903+	MOV	A, @R1				
0439 53F0	=2907	ANL	A, #0F0H				
	=2908	MORL	A, EPPCHI				
043B B925	=2914+	MOV	R1, #EPPCHI				
043D 41	=2915+	ORL	A, @R1				
043E 8A	=2919	MOV	LDATA, A				
043F F4C3	=2920	CALL	EPSTOR				
0441 B8D1	=2921 EPCNT:	MOV	R0, #LOW(OV2BAS+OV5IZE)				
0443 746A	=2922	CALL	OVLOAD				
	=2923	MMOV	A, EPR0				
0445 B923	=2932+	MOV	R1, #EPR0				
0447 F1	=2933+	MOV	A, @R1				
0448 F4D0	=2937	CALL	EPPASS				
	=2938	MMOV	A, EPPSW				
044A B921	=2947+	MOV	R1, #EPPSW				
044C F1	=2948+	MOV	A, @R1				
044D F4D0	=2952	CALL	EPPASS				
	=2953	MMOV	A, EPTIMR				
044F B922	=2962+	MOV	R1, #EPTIMR				
0451 F1	=2963+	MOV	A, @R1				
0452 F4D0	=2967	CALL	EPPASS				
	=2968	MMOV	A, EPACC				
0454 B920	=2977+	MOV	R1, #EPACC				
0456 F1	=2978+	MOV	A, @R1				
0457 F4D0	=2982	CALL	EPPASS				
0459 8903	=2983	ORL	P1, #00000011B				
045B F4D0	=2984	CALL	EPSTEP				
045D 745A	=2985	CALL	OVSAP				
045F 846B	=2986	JMP	CGO				
	=2987 ;						
	=2988 ;	COMGOR	GO FROM RESET COMMAND				
	=2989 ;		RESET PROCESSOR				
	=2990 ;		RELOAD LOW ORDER PROGRAM BYTES INTO PROGRAM MEMORY				
	=2991 ;						
0461 2302	=2992 COMGOR:	MOV	A, #2				
0463 3400	=2993	CALL	OUTUTL				
0465 8910	=2994	ORL	P1, #EPRSET				
0467 745A	=2995	CALL	OVSAP				
0469 99EF	=2996	ANL	P1, #(NOT EPRSET)				
	=2997 ;						
	=2998 ;						
	=2999 ;	CGO	SET UP BREAK LOGIC FOR APPROPRIATE BREAK CONDITIONS,				
	=3000 ;		DEPENDING ON CONTENTS OF 'TYPE'.				
	=3001 ;						
	=3002 CGO:	MMOV	A, TYPE				
046B B937	=3011+	MOV	R1, #TYPE				
046D F1	=3012+	MOV	A, @R1				
046E 0371	=3016	ADD	A, #LOW GOTEL				
0470 B3	=3017	JMPP	@A				
	=3018 ;						
0471 7C	=3019 GOTBL:	DB	LOW(CGONE)				

LOC	OBJ	LINE	SOURCE STATEMENT	THOUGHTS	LOC
0472	76	=3020	DB LOW(CGOMB)		
0473	80	=3021	DB LOW(CGOS5)		
0474	76	=3022	DB LOW(CGOPAT)		
0475	80	=3023	DB LOW(CGOTRA)		
		=3024 ;			
		=3025 CGOPAT:			
0476	99FD	=3026 CGOMB: ANL	P1.#NOT 00000010B		
0478	8901	=3027	ORL P1.#00000001B		
047A	8482	=3028	JMP EPRUN4		
		=3029 ;			
047C	99FC	=3030 CGOND: ANL	P1.#NOT 00000011B		
047E	8482	=3031	JMP EPRUN4		
		=3032 ;			
		=3033 CGOTRA:			
0480	8903	=3034 CGOS5: ORL	P1.#00000011B		
		=3035 ;			
		=3036 ; EPRUN4 SET UP CONTROL LOGIC TO RUN USER'S PROGRAM.			
		=3037 ;	RELEASE PROCESSOR TO RUN.		
		=3038 ;			
0482	8A20	=3039 EPRUN4: ORL	P2.#00100000B ; DISABLE EP LINK REFERENCES.		
0484	9AEE	=3040	ANL P2.#NOT 00010000B ; SET ALL REFERENCES TO RAM ARRAY.		
0486	99DF	=3041	ANL P1.#NOT MODOUT		
0488	F4F4	=3042	CALL EPREL		
		=3043 ;			
		=3044 ;	WAIT FOR KEYSTROKE INPUT OR HARDWARE BREAK TO OCCUR.		
		=3045 ;			
048A	F4AC	=3046 EPRUN1: CALL	TOFPOL		
048C	F4AF	=3047	CALL KBDPOL		
048E	37	=3048	CPL A		
048F	F295	=3049	JB7 EPRUN3		
0491	8699	=3050	JNI EPRUN2		
0493	848A	=3051	JMP EPRUN1		
		=3052 ;			
		=3053 ; EPRUN3 A KEYSTROKE WAS DETECTED WHILE EP WAS RUNNING.			
		=3054 ;	BREAK EXECUTION.		
		=3055 ;	PROCESS KEYSTROKE.		
0495	B400	=3056 EPRUN3: CALL	STSAVE		
0497	84B3	=3057	JMP EPRUN5		
		=3058 ;			
		=3059 ; EPRUN2 AN ENABLED BREAK CONDITION OCCURRED.			
		=3060 ;	BREAK EMULATION MODE.		
		=3061 ;	CONTINUE ACCORDING TO GO COMMAND TYPE.		
0499	B400	=3062 EPRUN2: CALL	STSAVE		
		=3063	MMOV A, TYPE		
049B	B937	=3072+	MOV RL, #TYPE		
049D	F1	=3073+	MOV A, @R1		
049E	03A1	=3077	ADD A, #LOW CNTTBL		
04A0	B3	=3078	JMPP @A		
		=3079 ;			
04A1	A6	=3080 CNTTBL: DB	LOW(BRKERR)		
04A2	BA	=3081	DB LOW(EPRUNG)		
04A3	BA	=3082	DB LOW(EPRUNG)		
04A4	AA	=3083	DB LOW(CNTTRA)		
04A5	AA	=3084	DB LOW(CNTTRA)		
		=3085 ;			

LOC	OBJ	LINE	SOURCE STATEMENT	TIME	LOC
		=3086 ;	BRKERR BREAKPOINT LATCH WAS SET THOUGH BREAKPOINTS NOT ENABLED.		
		=3087 ;	DISPLAY HARDWARE ERROR MESSAGE.		
04A6	BA0B	=3088	BRKERR: MOV LDATA, #0EH		
04A8	249A	=3089	JMP PERROR		
		=3090 ;			
		=3091	CNTR: MMOV A, DSPTIM		
04AA	D928	=3100+	MOV RL, #DSPTIM		
04AC	F1	=3101+	MOV A, @R1		
04AD	94F2	=3105	CALL DELAY		
04AF	F4F1	=3106	CALL KBDPOL		
04B1	F241	=3107	JB7 EPCNT ; B7 SET INDICATES NO KEYSTROKE.		
		=3108 ;			
		=3109 ;	EPRUN5 INPUT(KEY),		
		=3110 ;	IF KEY=END GO TO PARSER,		
		=3111 ;	INPUT KEY,		
		=3112 ;	IF KEY<NEXT GO TO PARSER,		
		=3113 ;	CONTINUE IN SAME MODE.		
		=3114 ;			
04B3	14EC	=3115	EPRUN5: CALL INPKEY		
04B5	FB	=3116	MOV A, KEY		
04B6	D313	=3117	XRL A, #KEYEND		
04B8	96C7	=3118	JNZ EPRET		
04BA	14EC	=3119	EPRUN6: CALL INPKEY		
04BC	FB	=3120	MOV A, KEY		
04BD	D312	=3121	XRL A, #KEYNXT		
04BF	96C7	=3122	JNZ EPRET		
04C1	2302	=3123	MOV A, #2		
04C3	3400	=3124	CALL OUTUTL		
04C5	8441	=3125	JMP EPCNT		
		=3126 ;			
		=3127 ;	EPRET EXECUTION MODE IS TO BE TERMINATED.		
		=3128 ;	JUMP INTO PARSER TO INTERPRET KEY ALREADY DETECTED.		
04C7	0433	=3129	EPRET: JMP MAIN2		
		=3130 ;			
		=3131	SIZECHK		
00C9		=3134+	SIZE SET 201		
		=3135+;			
		=3136+;	*****		
		=3145	\$EJECT		

LOC	OBJ	LINE	SOURCE STATEMENT	THIRTYTWO	THIRTY	THIRTY
		=3146	CODEBLK 115			
0500		=3171+	ORG 1200			
		=3175 ;	STSAVE EP STATUS SAVE SUBROUTINE.			
		=3176 ;	FORCE CALL TO LOC 014H;			
		=3177 ;	SAVE EP ACC;			
		=3178 ;	SAVE EP TIMER;			
		=3179 ;	SAVE EP PSW;			
		=3180 ;	SAVE EP R0;			
		=3181 ;	SAVE EP TOP-OF-STACK IN EP PC;			
		=3182 ;	RETURN.			
0500	744F	=3183	STSAVE: CALL EPBRK			
0502	2303	=3184	MOV R, #3			
0504	3400	=3185	CALL OUTUTL			
0506	745A	=3186	CALL OVSAMP			
0508	888F	=3187	MOV R0, #LOW(OV0BAS+OVSIZE)			
050A	746A	=3188	CALL OVLOAD			
050C	8A20	=3189	ORL P2, #00100000B			
050E	2314	=3190	MOV R, #14H			
0510	91	=3191	MOVX @R1, R			
0511	9ADF	=3192	ANL P2, #NOT 00100000B			
0513	8903	=3193	ORL P1, #00000011B			
0515	F4DB	=3194	CALL EPSTEP			
0517	8A20	=3195	ORL P2, #00100000B			
0519	9AEF	=3196	ANL P2, #NOT 00010000B			
051B	8903	=3197	ORL P1, #(ENBRAM OR ENBLNK)			
051D	F4DB	=3198	CALL EPSTEP			
		=3199 ;				
		=3200 ;	EXECUTION PROCESSOR IS NOW AT LOCATION 009H: INTERNAL WITH			
		=3201 ;	(RETURN ADDRESS+2) PUSHED ON STACK.			
		=3202 ;				
051F	88A5	=3203	MOV R0, #LOW(OV3BAS+OVSIZE)			
0521	746A	=3204	CALL OVLOAD			
0523	F4D0	=3205	CALL EPPASS			
		=3206	MMOV EPACC, R			
0525	B920	=3219+	MOV R1, #EPACC			
0527	A1	=3220+	MOV @R1, R			
0528	F4D0	=3224	CALL EPPASS			
		=3225	MMOV EPTIMR, R			
052A	B922	=3238+	MOV R1, #EPTIMR			
052C	A1	=3239+	MOV @R1, R			
052D	F4D0	=3243	CALL EPPASS			
		=3244	MMOV EPPSW, R			
052F	B921	=3257+	MOV R1, #EPPSW			
0531	A1	=3258+	MOV @R1, R			
0532	F4D0	=3262	CALL EPPASS			
		=3263	MMOV EPR0, R			
0534	B923	=3276+	MOV R1, #EPR0			
0536	A1	=3277+	MOV @R1, R			
0537	D8BB	=3281	MOV R0, #LOW(OV1BAS+OVSIZE)			
0539	746A	=3282	CALL OVLOAD			
		=3283	MMOV R, EPPSW			
053B	B921	=3292+	MOV R1, #EPPSW			
053D	F1	=3293+	MOV R, @R1			
053E	07	=3297	DEC R			
053F	5307	=3298	ANL R, #07H			

```

004+ 0930 =3314+ CMPOV CTOTR5 WSHLU
0546 A1 =3315+ MOV @R1, A
0547 F4B7 =3319 CALL EPFET
0549 03FE =3320 ADD A, #-2
054B AA =3321 MOV LDATA, A
=3322 MMOV EPPCLO, A
054C B924 =3335+ MOV R1, #EPPCLO
054E A1 =3336+ MOV @R1, A
054F F4C3 =3340 CALL EPSTOR
0551 B930 =3341 MOV R1, #SMALO
0553 11 =3342 INC @R1
0554 F4B7 =3343 CALL EPFET
0556 AA =3344 MOV LDATA, A
0557 53F0 =3345 ANL A, #11110000B
0559 2A =3346 XCH A, LDATA
055A 13FF =3347 ADDC A, #-1
055C 530F =3348 ANL A, #00001111B
=3349 MMOV EPPCHI, A
055E B925 =3362+ MOV R1, #EPPCHI
0560 A1 =3363+ MOV @R1, A
0561 4A =3367 ORL A, LDATA
0562 AA =3368 MOV LDATA, A
0563 F4C3 =3369 CALL EPSTOR
0565 B825 =3370 MOV R0, #EPPCHI
0567 347C =3371 CALL UPDAD1
0569 2340 =3372 MOV A, #01000000B ; "-" FOR DISPLAY
056B D4D8 =3373 CALL WDISP
056D B820 =3374 MOV R0, #EPACC
056F 3490 =3375 CALL DSPMID
0571 83 =3376 RET
=3377 SIZECHK
0072 =3380+ SIZE SET 114
=3381+;
=3382+; *****
=3391 $EJECT

```

```

SOURCE STATEMENT LINE
SAVE EP ACC 3314=
SAVE EP TOS 3315=
SAVE EP R0 3319=
SAVE EP R1 3320=
SAVE EP TOP-OF-STACK IN EP PC 3321=
RETURN 3322=
CALL EPFET 3335=
MOV A, A 3336=
CALL EPSTOR 3340=
CALL UPDAD1 3341=
MOV A, #11110000B 3342=
CALL EPFET 3343=
CALL LDATA 3344=
ANL A, #11110000B 3345=
XCH A, LDATA 3346=
ADDC A, #-1 3347=
ANL A, #00001111B 3348=
MMOV EPPCHI, A 3349=
CALL EPFET 3362=
MOV @R1, A 3363=
ORL A, LDATA 3367=
MOV LDATA, A 3368=
CALL EPSTOR 3369=
MOV R0, #EPPCHI 3370=
CALL UPDAD1 3371=
MOV A, #01000000B ; "-" FOR DISPLAY 3372=
CALL WDISP 3373=
MOV R0, #EPACC 3374=
CALL DSPMID 3375=
RET 3376=
SIZECHK 3377=
SIZE SET 114 3380+
3381+;
3382+; *****
3391 $EJECT

```

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOLS	ADDRESS
		3392 \$	INCLUDE(,F0:HFILE,MOD)		
000D		=3393	CHARCR EQU 0DH ;CCR		
000A		=3394	CHARLF EQU 0AH ;CLF		
001A		=3395	CNTRLZ EQU 1AH ;CONTROL-2		
		=3396			
		=3397	CODEBLK 80		
0297		=3412+	ORG 663		
		=3416	HRECIN HEXFILE RECORD INPUT ROUTINE		
0297 34CD		=3417	HRECIN: CALL CHARIN		
0299 D31A		=3418	XRL A,#CNTRLZ		
029B C6E0		=3419	JZ DONE		
029D D31A		=3420	XRL A,#CNTRLZ		
029F D33A		=3421	XRL A,#('')		
02A1 9697		=3422	JNZ HRECIN		
		=3423	MMOV CHKSUM,ZERO		
02A3 BD00		=3428+	MOV CHKSUM,#ZERO		
02A5 14F0		=3432	CALL BYTEIN		
		=3433	MMOV BUFNT,A		
02A7 B941		=3446+	MOV R1,#BUFNT		
02A9 A1		=3447+	MOV @R1,A		
02AA 14F0		=3451	CALL BYTEIN		
		=3452	MMOV SMARI,A		
02AC B931		=3465+	MOV R1,#SMARI		
02AE A1		=3466+	MOV @R1,A		
02AF 14F0		=3470	CALL BYTEIN		
		=3471	MMOV SMALO,A		
02B1 B930		=3484+	MOV R1,#SMALO		
02B3 A1		=3485+	MOV @R1,A		
02B4 14F0		=3489	CALL BYTEIN		
		=3490	MMOV RECTYP,A		
02B6 B942		=3503+	MOV R1,#RECTYP		
02B8 A1		=3504+	MOV @R1,A		
		=3508			
		=3509	HDATIN HEX DATA BYTE IN		
		=3510	HDATIN: MMOV A,BUFNT		
02B9 B941		=3519+	MOV R1,#BUFNT		
02BB F1		=3520+	MOV A,@R1		
02BC C6CC		=3524	JZ RECDON		
02BE 14F0		=3525	CALL BYTEIN		
02C0 AA		=3526	MOV LDATA,A		
02C1 F400		=3527	CALL LSTORE		
02C3 34F2		=3528	CALL INCSMA		
		=3529	MDEC BUFNT		
02C5 B941		=3534+	MOV R1,#BUFNT		
02C7 F1		=3535+	MOV A,@R1		
02C8 07		=3539+	DEC A		
02C9 A1		=3544+	MOV @R1,A		
02CA 44B9		=3547	JMP HDATIN		
		=3548			
02CC 34CD		=3549	RECDON: CALL CHARIN		
02CE D33F		=3550	XRL A,#('')		
02D0 C6DB		=3551	JZ CKSMOK		
02D2 D33F		=3552	XRL A,#('')		
02D4 34BA		=3553	CALL NIBIN2		
02D6 14F2		=3554	CALL BYTE11		

LOC	OBJ	LINE	SOURCE STATEMENT	THIRDS	STATE	LINE	LOC	OBJ
		=3555						
		=3556						
		=3557	MMOV A,CHKSUM					
02D8	FD	=3573+	MOV A,CHKSUM					
02D9	96E1	=3577	JNZ CHKERR					
		=3578	CKSMOK:MMOV A,RECTYP					
02DB	B942	=3587+	MOV R1,RECTYP					
02DD	F1	=3588+	MOV A,R1					
02DE	C697	=3592	JZ HRECIN					
		=3593 ;						
		=3594 ;	DONE HEX FILE CORRECTLY RECEIVED					
02E0	83	=3595	DONE: RET					
		=3596 ;						
		=3597 ;	CHKERR CHECKSUM ERROR IN INPUT RECORD DETECTED					
02E1	B80C	=3598	CHKERR: MOV LDAT, #0CH					
02E3	249A	=3599	JMP PLRERR					
		=3600	SIZECHK					
004E		=3603+	SIZE SET 78					
		=3604+;						
		=3605+;	*****					
		=3614 ;						
		=3615	CODEBLK 12					
00F0		=3620+	ORG 240					
		=3624 ;	BYTEIN BYTE INPUT SUBROUTINE.					
		=3625 ;	RECEIVES TWO HEXIDECIMAL CHARACTERS FROM THE TAPE INPUT DEVICE					
		=3626 ;	AND ASSEMBLES THEM INTO A SINGLE BYTE OF DATA					
00F0	34B8	=3627	BYTEIN: CALL NIBIN					
00F2	47	=3628	BYTEI1: SWAP A					
00F3	AA	=3629	MOV LDAT, A					
00F4	34B8	=3630	CALL NIBIN					
		=3631	MORL LDAT, A					
00F6	4A	=3640+	ORL A, LDAT					
00F7	AA	=3660+	MOV LDAT, A					
00F8	6D	=3664	ADD A, CHKSUM					
00F9	AD	=3665	MOV CHKSUM, A					
00FA	FA	=3666	MOV A, LDAT					
00FB	83	=3667	RET					
		=3668	SIZECHK					
006C		=3671+	SIZE SET 12					
		=3672+;						
		=3673+;	*****					
		=3682 ;						
		=3683	CODEBLK 25					
01B8		=3693+	ORG 440					
		=3697 ;	NIBIN RECEIVES A HEXIDECIMAL CHARACTER AND PRODUCES A MASKED FOUR BIT VALUE.					
		=3698 ;	NOTE- ERROR CHECKING DONE TO VERIFY HEXIDECIMAL VALIDITY					
01B8	34CD	=3699	NIBIN: CALL CHARIN					
01BA	03C6	=3700	NIBIN2: ADD A, #-3AH					
		=3701						
01BC	E6C2	=3702	JNC NIBI3					
01BE	03F9	=3703	ADD A, #-7					
01C0	E6C9	=3704	JNC RSCERR					
		=3705 ;						
		=3706 ;	ACC=0F61-05H FOR CHARACTERS '0'-'F'					
		=3707 ;						

LOC	OBJ	LINE	SOURCE STATEMENT	OBJECT STATEMENT	LINE	LOC	OBJ
01C2	03FA	=3708	NIBI3: ADD A, #=6 ; ACC=0F0H-0FFH FOR CHARACTERS '0'-'F'				
01C4	0310	=3709	ADD A, #10H ; ACC=00H-0FH FOR CHARACTERS '0'-'F';				
		=3710	OVERFLOW IF ABOVE IS TRUE.				
01C6	E6C9	=3711	JNC ASCERR				
01C8	83	=3712	RET				
		=3713	;				
		=3714	ASCERR ILLEGAL HEXIDECIMAL CHARACTER RECEIVED				
01C9	BA0A	=3715	ASCERR: MOV LDATA, #0AH				
01CB	249A	=3716	JMP PERROR				
		=3717	SIZECHK				
0015		=3720+	SIZE SET 21				
		=3721+	;				
		=3722+	*****				
		=3731	;				
		=3732	;				
		=3733	CODEBLK 5				
01CD		=3743+	ORG 461				
		=3747	CHARIN CHARACTER INPUT ROUTINE.				
		=3748	RECEIVES ONE ASCII CHARACTER FROM THE LOGICAL READER DEVICE.				
01CD	D449	=3749	CHARIN: CALL CIN				
01CF	537F	=3750	ANL A, #7FH				
01D1	83	=3751	RET				
		=3752	SIZECHK				
0005		=3755+	SIZE SET 5				
		=3756+	;				
		=3757+	*****				
		=3766	;				
		=3767	;				
		=3768	\$EJECT				


```

=3799 ; WHEN CALLED WITH F0=0 OUTPUT IS STANDARD HEX FILE FORMAT.
=3800 ; WHEN CALLED WITH F0=1 OUTPUT IS FORMATTED DATA DUMP TO CRT.
=3801 HFILE0: MOV MEMHI, SMAHI
0572 B931 =3817+ MOV R1, #SMAHI
0574 F1 =3818+ MOV R1, @R1
0575 B935 =3824+ MOV R1, #MEMHI
0577 A1 =3825+ MOV @R1, A
=3820 MOV MEMLO, SMALO
0578 B930 =3844+ MOV R1, #SMALO
057A F1 =3845+ MOV R1, @R1
057B B934 =3851+ MOV R1, #MEMLO
057D A1 =3852+ MOV @R1, A
=3855 MOV CHKSUM, ZERO
057E B000 =3860+ MOV CHKSUM, #ZERO
0580 B065 =3864 MOV R0, #HEXBUF
=3865 ;
=3866 ; LDBYTE LOAD NEXT BYTE FROM MEMORY INTO HEX BUFFER
0582 14FC =3867 LDBYTE: CALL LFETCH
0584 FA =3868 MOV A, LDATA
0585 A0 =3869 MOV @R0, A
0586 18 =3870 INC R0
0587 B4E2 =3871 CALL CMPHAS
0589 E696 =3872 JNC ENDFIL
058B 34F2 =3873 CALL INCSMA
058D F8 =3874 MOV A, R0
058E 038B =3875 ADD A, #- (BUFLN+HEXBUF)
0590 E682 =3876 JNC LDBYTE
0592 D400 =3877 CALL HREC0
0594 A472 =3878 JMP HFILE0
=3879 ;
=3880 ; ENDFIL END HEX FILE TRANSMISSION:
=3881 ; PRINT OUT BUFFER FOR LAST DATA RECORD
=3882 ; PRINT OUT CANNED 'END-OF-FILE' RECORD
=3883 ; RETURN
0596 D400 =3884 ENDFIL: CALL HREC0
0598 B6A7 =3885 JF0 HFDONE
059A 34D2 =3886 CALL TCRLFO
059C B8AE =3887 MOV R0, # (LOW EOFREC)
059E F8 =3888 ENDF1: MOV A, R0
059F A3 =3889 MOVP A, @A
05A0 C6A7 =3890 JZ HFDONE
05A2 B4BD =3891 CALL CHAR0
05A4 18 =3892 INC R0
05A5 A49E =3893 JMP ENDF1
05A7 34D2 =3894 HFDONE: CALL TCRLFO
05A9 231A =3895 MOV A, #CNTRLZ
05AB B4BD =3896 CALL CHAR0
05AD 83 =3897 RET
=3898 ;
=3899 ; EOFREC CHARACTER SKTING FOR CANNED END-OF-FILE RECORD FOR
=3900 ; INTEL HEX FILE FORMAT STANDARD.
05AE 203A3030 =3901 EOFREC: DB ' :00000001FF'

```

LOC	OBJ	LINE	SOURCE STATEMENT	THIRTYTWO	THIRTYTWO	THIRTYTWO	THIRTYTWO
05B2	30303030			A J08	MOV	+0200=	1A 3E00
05B6	30314646			DATA	MOV	+0000=	2E00 7E00
05BA	00	=3902	DB 0 ; END OF STRING CODE BYTE			+0000=	
		=3903	SIZECHK			+0000=	
0049		=3906+	SIZE SET 73			+0000=	
		=3907+				+0000=	
		=3908+	*****			+0000=	
		=3917 ;				+0000=	
		=3918 ;				+0000=	
		=3919	CODEBLK 90			+0000=	
0600		=3949+	ORG 1536			+0000=	
		=3953 ;	HRECO HEXIDECEMAL RECORD OUTPUT SEQUENCE.			+0000=	
		=3954 ;	HEX BUFFER ALREADY LOADED.			+0000=	
0600	F8	=3955	HRECO: MOV R0			+0000=	
0601	039B	=3956	ADD R, #HEXBUF			+0000=	
		=3957	MMOV BUFCONT, A			+0000=	
0603	B941	=3970+	MOV R1, #BUFCONT			+0000=	
0605	F1	=3971+	MOV @R1, A			+0000=	
0606	34D2	=3975	CALL TCRIFO			+0000=	
0608	2320	=3976	MOV A, #' '			+0000=	
060A	B4BD	=3977	CALL CHARO			+0000=	
060C	B617	=3978	JF0 FDUMP1			+0000=	
060E	233A	=3979	MOV A, #' '			+0000=	
0610	B4BD	=3980	CALL CHARO			+0000=	
		=3981	MMOV A, BUFCONT			+0000=	
0612	B941	=3990+	MOV R1, #BUFCONT			+0000=	
0614	F1	=3991+	MOV A, @R1			+0000=	
0615	34DB	=3995	CALL BYTEO			+0000=	
		=3996	FDUMP1: MMOV A, MEMHI			+0000=	
0617	B935	=4005+	MOV R1, #MEMHI			+0000=	
0619	F1	=4006+	MOV A, @R1			+0000=	
061A	34DB	=4010	CALL BYTEO			+0000=	
		=4011	MMOV A, MEMLO			+0000=	
061C	B934	=4020+	MOV R1, #MEMLO			+0000=	
061E	F1	=4021+	MOV A, @R1			+0000=	
061F	34DB	=4025	CALL BYTEO			+0000=	
0621	B628	=4026	JF0 FDUMP2			+0000=	
0623	27	=4027	CLR A			+0000=	
0624	34DB	=4028	CALL BYTEO			+0000=	
0626	C42C	=4029	JMP DATO			+0000=	
0628	233D	=4030	FDUMP2: MOV A, #' '			+0000=	
062A	B4BD	=4031	CALL CHARO			+0000=	
		=4032 ;	DATO DATA OUTPUT			+0000=	
062C	B865	=4033	DATO: MOV R0, #HEXBUF			+0000=	
062E	B632	=4034	DATO1: JF0 FDUMP5			+0000=	
0630	C436	=4035	JMP FDUMP3			+0000=	
0632	2320	=4036	FDUMP5: MOV A, #' '			+0000=	
0634	B4BD	=4037	CALL CHARO			+0000=	
0636	F0	=4038	FDUMP3: MOV A, @R0			+0000=	
0637	34DB	=4039	CALL BYTEO			+0000=	
0639	18	=4040	INC R0			+0000=	
		=4041	MDJNZ BUFCONT, DATO1			+0000=	
063A	B941	=4046+	MOV R1, #BUFCONT			+0000=	
063C	F1	=4047+	MOV A, @R1			+0000=	
063D	07	=4051+	DEC A			+0000=	

LOC	OBJ	LINE	SOURCE STATEMENT	STATEMENT	LINE	LOC	OBJ
063E	R1	=4056+	MOV @R1, A			063E	R1
063F	962E	=4060+	JNZ DAT01			063F	962E
		=4062 ;					
		=4063 ;	ENDREC END RECORD BEING TRANSMITTED				
0641	B648	=4064	ENDREC: JF0 FDUMP4			0641	B648
		=4065	MMOV A, CHKSUM				
0643	FD	=4081+	MOV A, CHKSUM			0643	FD
0644	37	=4085	CPL A			0644	37
0645	17	=4086	INC A			0645	17
0646	34DB	=4087	CALL BYTE0			0646	34DB
0648	83	=4088	FDUMP4: RET			0648	83
		=4089	SIZECHK				
0049		=4092+	SIZE SET 73			0049	
		=4093+;					
		=4094+;	*****				
		=4103 ;					
		=4104	CODEBLK 9				
01D2		=4114+	ORG 466			01D2	
		=4118 ;	TCRLF0 TAPE <CR><LF> OUTPUT				
01D2	230D	=4119	TCRLF0: MOV A, #CHARCR			01D2	230D
01D4	B4BD	=4120	CALL CHAR0			01D4	B4BD
01D6	230A	=4121	MOV A, #CHARLF			01D6	230A
01D8	B4BD	=4122	CALL CHAR0			01D8	B4BD
01DA	83	=4123	RET			01DA	83
		=4124	SIZECHK				
0009		=4127+	SIZE SET 9			0009	
		=4128+;					
		=4129+;	*****				
		=4138 ;					
		=4139	CODEBLK 11				
01D8		=4149+	ORG 475			01D8	
		=4153 ;	BYTE0 BYTE OUTPUT				
01D8	AA	=4154	BYTE0: MOV LDATA, A			01D8	AA
01DC	6D	=4155	ADD A, CHKSUM			01DC	6D
01DD	AD	=4156	MOV CHKSUM, A			01DD	AD
01DE	FA	=4157	MOV A, LDATA			01DE	FA
01DF	47	=4158	SWAP A			01DF	47
01E0	B4DB	=4159	CALL NIB0			01E0	B4DB
01E2	FA	=4160	MOV A, LDATA			01E2	FA
01E3	B4DB	=4161	CALL NIB0			01E3	B4DB
01E5	83	=4162	RET			01E5	83
		=4163	SIZECHK				
0008		=4166+	SIZE SET 11			0008	
		=4167+;					
		=4168+;	*****				
		=4177 ;					
		=4178	CODEBLK 12				
01E6		=4188+	ORG 486			01E6	
		=4192 ;	HEXASC HEXIDECIMAL NIBBLE TO ASCII CHARACTER CONVERSION				
01E6	530F	=4193	HEXASC: ANL A, #0FH			01E6	530F
01E8	03F6	=4194	ADD A, #(-10)			01E8	03F6
01EA	F6EF	=4195	JC HEXNIB			01EA	F6EF
01EC	033A	=4196	ADD A, #(-10+'0')			01EC	033A
01EE	83	=4197	RET			01EE	83
01EF	0341	=4198	HEXNIB: ADD A, #('A')			01EF	0341

LOC	OBJ	LINE	SOURCE STATEMENT
01F1	83	=4199	RET
		=4200	SIZECHK
000C		=4203+ SIZE SET 12	
		=4204+;	
		=4205+; *****	
		=4214 ;	
		=4215 ;	
		=4216 DECLARE BIT50, CONST	
000B		=4230 BIT50 EQU 11 ; DATA BITS PUT OUT (INCLUDING TWO STOP BITS)	
		=4231 ;	
		=4232 CODEBLK 30	
04C9		=4252+ ORG 1225	
		=4256 ; HBDLAY HALF-BIT TIME DELAY	
		=4257 HBDLAY: MMOV H, HBITHI	
04C9 B927		=4273+ MOV R1, #HBITHI	
04CB F1		=4274+ MOV R, @R1	
04CC B945		=4280+ MOV R1, #H	
04CE A1		=4281+ MOV @R1, A	
		=4284 MMOV R1, HBITLO	
04CF B926		=4300+ MOV R1, #HBITLO	
04D1 F1		=4301+ MOV R, @R1	
04D2 A9		=4314+ MOV R1, A	
04D3 84D7		=4317 JMP HBD1	
04D5 B900		=4318 HBD2: MOV R1, #0	
04D7 E9D7		=4319 HBD1: DJNZ R1, HBD1	
		=4320 MDJNZ H, HBD2	
04D9 B945		=4325+ MOV R1, #H	
04DB F1		=4326+ MOV R, @R1	
04DC 07		=4330+ DEC A	
04DD A1		=4335+ MOV @R1, A	
04DE 96D5		=4339+ JNZ HBD2	
04E0 83		=4341 RET	
		=4342 SIZECHK	
0018		=4345+ SIZE SET 24	
		=4346+;	
		=4347+; *****	
		=4356 ;	
		=4357 \$EJECT	

```

=4387 ;NIBO MASK ACC TO MAKE HEX NIBBLE, TRANSLATE TO ASCII AND OUTPUT
05BB 34E6 =4388 NIBO: CALL HEXASC
=4389 ;
=4390 ;CHARO CONSOLE OUTPUT SUBROUTINE
=4391 ; WRITES THE CONTENTS OF THE ACC TO THE CRT DISPLAY SCREEN
=4392 CHARO: MMOV REGC, A
05BD B944 =4405+ MOV R1, #REGC
05BF A1 =4406+ MOV @R1, A
=4410 MMOV B, BIT50 ;SET NUMBER OF BITS TO BE TRANSMITTED
05C0 B943 =4421+ MOV R1, #B
05C2 B108 =4422+ MOV @R1, #BIT50
05C4 97 =4426 CLR C ;CLEAR CARRY
05C5 F6CB =4427 C01: JC C02
05C7 99BF =4428 ANL P1, #NOT TTYOUT
05C9 A4CF =4429 JMP C03
05CB 8940 =4430 C02: ORL P1, #TTYOUT
05CD 00 =4431 NOP ;EVEN OUT TWO BRANCH EXECUTION TIMES
05CE 00 =4432 NOP
05CF 94C9 =4433 C03: CALL HBDLAY
05D1 94C9 =4434 CALL HBDLAY
05D3 97 =4435 CLR C ;SET WHAT WILL EVENTUALLY BECOME A STOP BIT
05D4 A7 =4436 CPL C
=4437 MRRC REGC ;ROTATE CHARACTER RIGHT ONE BIT,
05D5 B944 =4442+ MOV R1, #REGC
05D7 F1 =4443+ MOV A, @R1
05D8 67 =4444+ RRC A
05D9 A1 =4452+ MOV @R1, A
=4455 ;\ MOVING NEXT DATA BIT INTO CARRY
=4456 MDJNZ B, C01 ;CHECK IF CHARACTER (AND STOP BIT(S)) DONE
05DA B943 =4461+ MOV R1, #B
05DC F1 =4462+ MOV A, @R1
05DD 07 =4466+ DEC A
05DE A1 =4471+ MOV @R1, A
05DF 96C5 =4475+ JNZ C01
05E1 83 =4477 RET
=4478 SIZECHK
0027 =4481+ SIZE SET 39
=4482+;
=4483+;*****
=4492 ;
=4493 CODEBLK 47
0649 =4523+ ORG 1609
=4527 ;CIN CONSOLE INPUT SUBROUTINE WAITS FOR A KEYSTROKE AND
=4528 ; RETURNS WITH 8 BITS IN REG ACC.
0649 B943 =4529 CIN: MOV R1, #B
064B B108 =4530 MOV @R1, #C ;DATA BITS TO BE READ
064D 464D =4531 CI0: JNT1 CI0
064F 464D =4532 JNT1 CI0
0651 5651 =4533 CI1: JT1 CI1
0653 5651 =4534 JT1 CI1
0655 94C9 =4535 CALL HBDLAY
0657 5651 =4536 JT1 CI1
0659 94C9 =4537 CI2: CALL HBDLAY

```


LOC	OBJ	LINE	SOURCE STATEMENT	OBJECT STATEMENT	LOC
065B	94C9	=4538	CALL HBDLAY	CALL HBDLAY	065B
065D	5662	=4539	JT1 CI3 ;CHECK SID LINE LEVEL	JT1 CI3 ;CHECK SID LINE LEVEL	065D
065F	97	=4540	CLR C ;DATA BIT IN CY	CLR C ;DATA BIT IN CY	065F
0660	C465	=4541	JMP CI4	JMP CI4	0660
0662	97	=4542	CI3: CLR C	CI3: CLR C	0662
0663	A7	=4543	CPL C	CPL C	0663
0664	00	=4544	NOP ;EVEN OUT BRANCH EXECUTION TIMES	NOP ;EVEN OUT BRANCH EXECUTION TIMES	0664
0665	00	=4545	CI4: NOP	CI4: NOP	0665
0666	00	=4546	NOP	NOP	0666
0667	00	=4547	NOP	NOP	0667
		=4548	MRRC REGC	MRRC REGC	
0668	B944	=4553+	MOV R1, #REGC	MOV R1, #REGC	0668
066A	F1	=4554+	MOV A, @R1	MOV A, @R1	066A
066B	67	=4558+	RRC A	RRC A	066B
066C	A1	=4563+	MOV @R1, A	MOV @R1, A	066C
		=4566	MDJNZ B, CI2	MDJNZ B, CI2	
066D	B943	=4571+	MOV R1, #B	MOV R1, #B	066D
066F	F1	=4572+	MOV A, @R1	MOV A, @R1	066F
0670	07	=4576+	DEC A	DEC A	0670
0671	A1	=4581+	MOV @R1, A	MOV @R1, A	0671
0672	9659	=4585+	JNZ CI2	JNZ CI2	0672
		=4587	MMOV A, REGC	MMOV A, REGC	
0674	B944	=4596+	MOV R1, #REGC	MOV R1, #REGC	0674
0676	F1	=4597+	MOV A, @R1	MOV A, @R1	0676
0677	83	=4601	RET ;CHARACTER COMPLETE	RET ;CHARACTER COMPLETE	0677
		=4602	SIZECHK	SIZECHK	
002F		=4605+	SIZE SET 47	SIZE SET 47	002F
		=4606+	*****	*****	
		=4607+	*****	*****	
		=4616	\$EJECT	\$EJECT	

LOC	OBJ	LINE	SOURCE STATEMENT	THOMAS2 SOURCE	LINE	LOC
		4617 \$	INCLUDE(:F0:MEMREF.MOD)			
		=4618	CODEBLK 15			
02E5		=4633+	ORG 741			
		=4637 ;	CONFIL COMMAND TO FILL ADDRESS SPACE BETWEEN SMA AND EMA WITH DATA			
		=4638 ;	IN LOW BYTE OF MEM.			
		=4639 CONFIL:	MNOV LDATA, MEMLO			
02E5 B934		=4635+	MOV R1, #MEMLO			
02E7 F1		=4656+	MOV A, @R1			
02E8 AA		=4669+	MOV LDATA, A			
02E9 F400		=4672 LFILL:	CALL LSTORE			
02EB B4E2		=4673	CALL CMPLAS			
02ED E6F3		=4674	JNC LFILL1			
02EF 34F2		=4675	CALL INCSMA			
02F1 44E9		=4676	JMP LFILL			
02F3 83		=4677 LFILL1:	RET			
		=4678	SIZECHK			
000F		=4681+	SIZE SET 15			
		=4682+;				
		=4683+;	*****			
		=4692 ;				
		=4693	CODEBLK 4			
00FC		=4698+	ORG 252			
		=4702 ;	LFETCH FETCHES CONTENTS OF LOGICAL MEMORY ADDRESS DETERMINED BY			
		=4703 ;	<TYPE>, <SMAHI>, & <SMALO> INTO <LDATA>.			
00FC D478		=4704 LFETCH:	CALL AFETCH			
00FE AA		=4705	MOV LDATA, A			
00FF 83		=4706	RET			
		=4707	SIZECHK			
0004		=4710+ SIZE SET 4				
		=4711+;				
		=4712+;	*****			
		=4721 ;				
		=4722	CODEBLK 75			
0678		=4752+	ORG 1656			
		=4756 ;				
		=4757 ;	AFETCH LOGICAL FETCH SUBROUTINE			
		=4758 ;	FETCHS CONTENTS OF VARIOUS MEMORY SPACES TO ACC.			
		=4759 AFETCH:	MNOV A, TYPE			
0678 B937		=4768+	MOV R1, #TYPE			
067A F1		=4769+	MOV A, @R1			
067B 037E		=4773	ADD A, #LOW LFETBL			
067D B3		=4774	JMPP @A			
		=4775 ;				
067E 04		=4776 LFETBL:	DB LOW LFEPM			
067F 98		=4777	DB LOW LFEDM			
0680 9C		=4778	DB LOW LFEREG			
0681 A9		=4779	DB LOW LFEINT			
0682 B1		=4780	DB LOW LFEBRK			
0683 B1		=4781	DB LOW LFEBRK			
		=4782 ;				
		=4783 LFEPM:	MNOV A, SMAHI			
0684 B931		=4792+	MOV R1, #SMAHI			
0686 F1		=4793+	MOV A, @R1			
0687 9698		=4797	JNZ LFEDM			
		=4798	MNOV A, SMALO			

LOC	OBJ	LINE	SOURCE STATEMENT	THIRGATWZ ECRUG2	THE	LOC	OBJ
0689	B930	=4807+	MOV R1, #SMALO	CODEBK 85	=4812		
068C	F1	=4808+	MOV A, @R1	0000 1325	=4813		
068C	03E9	=4812	ADD A, #-OVSZ		=4814		
068E	F698	=4813	JC LFEDM	ENTRANUS 3502 3000J 3001	=4815		
		=4814	MNOV A, SMALO	STORE LOGICAL STORE SUBROUTINE	=4816		
0690	B930	=4823+	MOV R1, #SMALO	R TYPE	=4817		
0692	F1	=4824+	MOV A, @R1	MOV R1 TYPE	=4818		
0693	034E	=4820	ADD A, #OVBUF	MOV A, #0	=4819		
0695	A9	=4829	MOV R1, A	MOV A, #0	=4820		
0696	F1	=4830	MOV A, @R1	MOV A, #0	=4821		
0697	83	=4831	RET		=4822		
0698	94E1	=4832	LFEDM: CALL LPGSEL	LOW LSTN	=4823		
069A	81	=4833	MOVX A, @R1	LOW LSTN	=4824		
069B	83	=4834	RET	LOW LSTN	=4825		
		=4835 ;		LOW LSTN	=4826		
		=4836	LFEREG: MNOV A, SMALO	LOW LSTN	=4827		
069C	B930	=4845+	MOV R1, #SMALO	LOW LSTN	=4828		
069E	F1	=4846+	MOV A, @R1		=4829		
069F	537F	=4850	ANL A, #01111111B ;CHECK IF LOW 7 BITS =0	LOW LSTN	=4830		
06A1	C6R5	=4851	JZ LFER0	LOW LSTN	=4831		
06A3	E4B7	=4852	JMP EPFET	LOW LSTN	=4832		
		=4853 ;		LOW LSTN	=4833		
		=4854	LFER0: MNOV A, EPFR0	LOW LSTN	=4834		
06A5	B923	=4863+	MOV R1, #EPFR0	LOW LSTN	=4835		
06A7	F1	=4864+	MOV A, @R1	LOW LSTN	=4836		
06A8	83	=4868	RET	LOW LSTN	=4837		
		=4869 ;		LOW LSTN	=4838		
		=4870	LFEINT: MNOV A, SMALO	LOW LSTN	=4839		
06A9	B930	=4879+	MOV R1, #SMALO	LOW LSTN	=4840		
06AB	F1	=4880+	MOV A, @R1	LOW LSTN	=4841		
06AC	0320	=4884	ADD A, #EPACC	LOW LSTN	=4842		
06AE	A9	=4885	MOV R1, A	LOW LSTN	=4843		
06AF	F1	=4886	MOV A, @R1	LOW LSTN	=4844		
06B0	83	=4887	RET	LOW LSTN	=4845		
		=4888 ;		LOW LSTN	=4846		
		=4889	LFEBRK: LOGICAL FETCH OF BREAK-POINT DATA	LOW LSTN	=4847		
06B1	94E1	=4890	LFEBRK: CALL LPGSEL	LOW LSTN	=4848		
06B3	99F7	=4891	ANL P1, #NOT 00001000B	LOW LSTN	=4849		
06B5	8908	=4892	ORL P1, #00001000B	LOW LSTN	=4850		
06B7	99FD	=4893	ANL P1, #NOT 00000010B	LOW LSTN	=4851		
06B9	8901	=4894	ORL P1, #00000001B	LOW LSTN	=4852		
06BB	81	=4895	MOVX A, @R1	LOW LSTN	=4853		
06BC	2301	=4896	MOV A, #01H	LOW LSTN	=4854		
06BE	86C1	=4897	JNI LFEBR1	LOW LSTN	=4855		
06C0	27	=4898	CLR A	LOW LSTN	=4856		
06C1	83	=4899	LFEBR1: RET	LOW LSTN	=4857		
		=4900	SIZECHK	LOW LSTN	=4858		
004A		=4903+	SIZE SET 74	LOW LSTN	=4859		
		=4904+;		LOW LSTN	=4860		
		=4905+;	*****	LOW LSTN	=4861		
		=4914	\$EJECT	LOW LSTN	=4862		

0700	=4950+	ORG	1792				
	=4954 ;						
	=4955 ;	LSTORE	LOGICAL STORE SUBROUTINE				
	=4956 ;		STORES CONTENTS OF LDATA INTO VARIOUS MEMORY SPACES.				
	=4957 ;	LSTORE:	MMOV A, TYPE				
0700 B937	=4966+	MOV	R1, #TYPE				
0702 F1	=4967+	MOV	A, @R1				
0703 0306	=4971	ADD	A, #LOW LSTBL				
0705 B3	=4972	JMPP	@A				
	=4973 ;						
0706 0C	=4974	LSTBL:	DB LOW LSTPM				
0707 21	=4975	DB	LOW LSTDM				
0708 26	=4976	DB	LOW LSTREG				
0709 34	=4977	DB	LOW LSTINT				
070H 3D	=4978	DB	LOW LSTBRK				
0700 3D	=4979	DB	LOW LSTBRK				
	=4980 ;						
	=4981 ;	LSTPM:	MMOV A, #SMALI				
070C B931	=4990+	MOV	R1, #SMALI				
070E F1	=4991+	MOV	A, @R1				
070F 9621	=4995	JNZ	LSTDM				
	=4996	MMOV	A, #SMALO				
0711 B930	=5005+	MOV	R1, #SMALO				
0713 F1	=5006+	MOV	A, @R1				
0714 03E9	=5010	ADD	A, #OVSZIE				
0716 F621	=5011	JC	LSTDM				
	=5012	MMOV	A, #SMALO				
0718 B930	=5021+	MOV	R1, #SMALO				
071A F1	=5022+	MOV	A, @R1				
071B 034E	=5026	ADD	A, #OVBIF				
071D A9	=5027	MOV	R1, A				
071E FA	=5028	MOV	A, LDATA				
071F A1	=5029	MOV	@R1, A				
0720 83	=5030	RET					
	=5031 ;						
0721 94E1	=5032	LSTDM:	CALL LPSSEL				
0723 FA	=5033	MOV	A, LDATA				
0724 91	=5034	MOVX	@R1, A				
0725 83	=5035	RET					
	=5036 ;						
	=5037 ;	LSTREG:	MMOV A, #SMALO				
0726 B930	=5046+	MOV	R1, #SMALO				
0728 F1	=5047+	MOV	A, @R1				
0729 537F	=5051	ANL	A, #01111111B ; CHECK IF LOW ORDER BITS = 0				
072B C62F	=5052	JZ	LSTR0				
072D E4C3	=5053	JMP	EPSTOR				
	=5054 ;						
	=5055 ;	LSTR0:	MMOV EP0, LDATA				
072F FA	=5078+	MOV	A, LDATA				
0730 B923	=5084+	MOV	R1, #EP0				
0732 A1	=5085+	MOV	@R1, A				
0733 83	=5088	RET					
	=5089 ;						
	=5090 ;	LSTINT:	MMOV A, #SMALO				

LOC	OBJ	LINE	SOURCE STATEMENT	THIRTHAT2 30A002	SWLJ	LOC	OBJ
0734	B930	=5099+	MOV R1, #SMALO	00000000	00000000	0734	B930
0736	F1	=5100+	MOV A, @R1	00000000	00000000	0736	F1
0737	0320	=5104	ADD A, #EPACC	00000000	00000000	0737	0320
0739	A9	=5105	MOV R1, A	00000000	00000000	0739	A9
073A	FA	=5106	MOV A, LDATA	00000000	00000000	073A	FA
073B	A1	=5107	MOV @R1, A	00000000	00000000	073B	A1
073C	83	=5108	RET	00000000	00000000	073C	83
		=5109 ;					
		=5110 ; LSTBRK LOGICAL STORE OF BREAK-POINT DATA					
073D	94E1	=5111	LSTBRK: CALL LPGSEL	00000000	00000000	073D	94E1
073F	FA	=5112	MOV A, LDATA	00000000	00000000	073F	FA
0740	1246	=5113	JB0 LSTBR1	00000000	00000000	0740	1246
0742	8901	=5114	ORL P1, #00000001B	00000000	00000000	0742	8901
0744	E448	=5115	JMP LSTBR2	00000000	00000000	0744	E448
0746	99FE	=5116	LSTBR1: ANL P1, #NOT 00000001B	00000000	00000000	0746	99FE
0748	99F7	=5117	LSTBR2: ANL P1, #NOT 00001000B	00000000	00000000	0748	99F7
074A	81	=5118	MOVX A, @R1	00000000	00000000	074A	81
074B	8908	=5119	ORL P1, #00001000B	00000000	00000000	074B	8908
074D	83	=5120	RET	00000000	00000000	074D	83
		=5121	SIZECHK				
004E		=5124+ SIZE SET 78					
		=5125+;					
		=5126+; *****					
		=5135 ;					
		=5136 CODEBLK 17					
04E1		=5156+ ORG 1249					
		=5160 ; LPGSEL LOGICAL PAGE SELECT.					
		=5161 ; SETS UP PORT 2 TO ADDRESS APPROPRIATE BYTE OF RAM BLOCK.					
		=5162 LPGSEL: MMOV A, TYPE					
04E1	B937	=5171+	MOV R1, #TYPE	00000000	00000000	04E1	B937
04E3	F1	=5172+	MOV A, @R1	00000000	00000000	04E3	F1
04E4	5301	=5176	ANL A, #00000001B ; MASK OFF DATA TYPE SELECTOR BIT	00000000	00000000	04E4	5301
04E6	47	=5177	SWAP A	00000000	00000000	04E6	47
		=5178	MORL A, #MAHI	00000000	00000000		
04E7	B931	=5104+	MOV R1, #MAHI	00000000	00000000	04E7	B931
04E9	41	=5185+	ORL A, @R1	00000000	00000000	04E9	41
04E8	4340	=5109	ORL A, #01000000B	00000000	00000000	04E8	4340
04EC	3A	=5190	OUTL P2, A	00000000	00000000	04EC	3A
		=5191	MMOV A, #SMALO	00000000	00000000		
04ED	B930	=5200+	MOV R1, #SMALO	00000000	00000000	04ED	B930
04EF	F1	=5201+	MOV A, @R1	00000000	00000000	04EF	F1
04F0	A9	=5205	MOV R1, A	00000000	00000000	04F0	A9
04F1	83	=5206	RET	00000000	00000000	04F1	83
		=5207	SIZECHK				
0011		=5210+ SIZE SET 17					
		=5211+;					
		=5212+; *****					
		=5221 ;					
		=5222 \$EJECT					

LOC	OBJ	LINE	SOURCE STATEMENT	THIRDS	CONV	LINE	LOC
		=5223	CODEBLK 11				
01F2		=5233+	ORG 498				
		=5237 ; INCSMA	INCREMENT STARTING MEMORY ADDRESS WORD.				
01F2 B930		=5238	INCSMA: MOV R1, #SMALO				
01F4 11		=5239	INCH: INC R1				
01F5 F1		=5240	MOV A, R1				
01F6 96FC		=5241	JNZ INCH1				
01F8 19		=5242	INC R1				
01F9 F1		=5243	MOV A, R1				
01FA 17		=5244	INC A				
01FB 31		=5245	XCHD A, R1				
01FC 83		=5246	INCH1: RET				
		=5247	SIZECHK				
000B		=5250+	SIZE SET 11				
		=5251+;					
		=5252+;	*****				
		=5261 ;					
		=5262	CODEBLK 12				
02F4		=5277+	ORG 756				
		=5281 ; DECSMA	DECREMENT SMA WORD.				
02F4 B930		=5282	DECSMA: MOV R1, #SMALO				
02F6 F1		=5283	MOV A, R1				
02F7 07		=5284	DEC A				
02F8 21		=5285	XCH A, R1				
02F9 96FF		=5286	JNZ DECSM1				
02FB 19		=5287	INC R1				
02FC F1		=5288	MOV A, R1				
02FD 07		=5289	DEC A				
02FE 31		=5290	XCHD A, R1				
02FF 83		=5291	DECSM1: RET				
		=5292	SIZECHK				
000C		=5295+	SIZE SET 12				
		=5296+;					
		=5297+;	*****				
		=5306 ;					
		=5307	CODEBLK 15				
05E2		=5332+	ORG 1506				
		=5336 ; CMPMAS	COMPARE MEMORY ADDRESSES				
		=5337 ;	COMPARE SMA BYTES WITH EMA BYTES TO DETERMINE RELATIVE MAGNITUDE.				
		=5338 ;	RETURNS WITH CARRY=1 IFF <SMA> >= <EMA>.				
		=5339 ;	IS CALLED AFTER ACTION HAS BEEN PERFORMED ON <SMA> TO DETERMINE IF				
		=5340 ;	TASK IS COMPLETED:				
		=5341 ;	IF CY=0 THEN <SMA> >= <EMA> ==> TERMINATE TASK.				
		=5342 ;	IF CY=1 THEN <SMA> < <EMA> ==> INC SMA AND REPEAT.				
		=5343	CMPMAS: MMOV A, SMALO				
05E2 B930		=5352+	MOV R1, #SMALO				
05E4 F1		=5353+	MOV A, R1				
05E5 37		=5357	CPL A				
		=5358	MADD A, EMALO				
05E6 B932		=5364+	MOV R1, #EMALO				
05E8 61		=5365+	ADD A, R1				
		=5369	MMOV A, SMAHI				
05E9 B931		=5378+	MOV R1, #SMAHI				
05EB F1		=5379+	MOV A, R1				
05EC 37		=5383	CPL A				

LOC	OBJ	LINE	SOURCE STATEMENT	THIRDS STATEMENT	LINE	LOC	OBJ
		=5384	MADDC A,EMAH1	(CONV: P0:3000)	2	05ED	B933
		=5390+	MOV R1,EMAH1	CONV: P0:3000	=5415+	05EF	71
		=5391+	ADDC A,R1	CONV: P0:3000	=5416+	05F0	83
		=5395	CMPT: RET		=5417		
		=5396	SIZECHK	KEYBOARD AND DISPLAY PROCESSING ROUTINE	=5418		
		=5399+	SIZE SET 15	ORLED PERIODICALLY WHEN KEY AND DISPLAY ARE IN	=5419		
		=5400+		SET	=5420		
		=5401+;	*****	*****	=5421		
		=5410	\$EJECT	MOV R1,R1	=5422		
				MOV R1,R1	=5423		
				MOV R1,R1	=5424		
				MOV R1,R1	=5425		
				MOV R1,R1	=5426		
				MOV R1,R1	=5427		
				MOV R1,R1	=5428		
				MOV R1,R1	=5429		
				MOV R1,R1	=5430		
				MOV R1,R1	=5431		
				MOV R1,R1	=5432		
				MOV R1,R1	=5433		
				MOV R1,R1	=5434		
				MOV R1,R1	=5435		
				MOV R1,R1	=5436		
				MOV R1,R1	=5437		
				MOV R1,R1	=5438		
				MOV R1,R1	=5439		
				MOV R1,R1	=5440		
				MOV R1,R1	=5441		
				MOV R1,R1	=5442		
				MOV R1,R1	=5443		
				MOV R1,R1	=5444		
				MOV R1,R1	=5445		
				MOV R1,R1	=5446		
				MOV R1,R1	=5447		
				MOV R1,R1	=5448		
				MOV R1,R1	=5449		
				MOV R1,R1	=5450		
				MOV R1,R1	=5451		
				MOV R1,R1	=5452		
				MOV R1,R1	=5453		
				MOV R1,R1	=5454		
				MOV R1,R1	=5455		
				MOV R1,R1	=5456		
				MOV R1,R1	=5457		
				MOV R1,R1	=5458		
				MOV R1,R1	=5459		
				MOV R1,R1	=5460		
				MOV R1,R1	=5461		
				MOV R1,R1	=5462		
				MOV R1,R1	=5463		
				MOV R1,R1	=5464		
				MOV R1,R1	=5465		
				MOV R1,R1	=5466		
				MOV R1,R1	=5467		
				MOV R1,R1	=5468		
				MOV R1,R1	=5469		
				MOV R1,R1	=5470		
				MOV R1,R1	=5471		
				MOV R1,R1	=5472		
				MOV R1,R1	=5473		
				MOV R1,R1	=5474		
				MOV R1,R1	=5475		
				MOV R1,R1	=5476		
				MOV R1,R1	=5477		
				MOV R1,R1	=5478		
				MOV R1,R1	=5479		
				MOV R1,R1	=5480		
				MOV R1,R1	=5481		
				MOV R1,R1	=5482		
				MOV R1,R1	=5483		
				MOV R1,R1	=5484		
				MOV R1,R1	=5485		
				MOV R1,R1	=5486		
				MOV R1,R1	=5487		
				MOV R1,R1	=5488		
				MOV R1,R1	=5489		
				MOV R1,R1	=5490		
				MOV R1,R1	=5491		
				MOV R1,R1	=5492		
				MOV R1,R1	=5493		
				MOV R1,R1	=5494		
				MOV R1,R1	=5495		
				MOV R1,R1	=5496		
				MOV R1,R1	=5497		
				MOV R1,R1	=5498		
				MOV R1,R1	=5499		
				MOV R1,R1	=5500		
				MOV R1,R1	=5501		
				MOV R1,R1	=5502		
				MOV R1,R1	=5503		
				MOV R1,R1	=5504		
				MOV R1,R1	=5505		
				MOV R1,R1	=5506		
				MOV R1,R1	=5507		
				MOV R1,R1	=5508		
				MOV R1,R1	=5509		
				MOV R1,R1	=5510		
				MOV R1,R1	=5511		
				MOV R1,R1	=5512		
				MOV R1,R1	=5513		
				MOV R1,R1	=5514		
				MOV R1,R1	=5515		
				MOV R1,R1	=5516		
				MOV R1,R1	=5517		
				MOV R1,R1	=5518		
				MOV R1,R1	=5519		
				MOV R1,R1	=5520		
				MOV R1,R1	=5521		
				MOV R1,R1	=5522		
				MOV R1,R1	=5523		
				MOV R1,R1	=5524		
				MOV R1,R1	=5525		
				MOV R1,R1	=5526		
				MOV R1,R1	=5527		
				MOV R1,R1	=5528		
				MOV R1,R1	=5529		
				MOV R1,R1	=5530		
				MOV R1,R1	=5531		
				MOV R1,R1	=5532		
				MOV R1,R1	=5533		
				MOV R1,R1	=5534		
				MOV R1,R1	=5535		
				MOV R1,R1	=5536		
				MOV R1,R1	=5537		
				MOV R1,R1	=5538		
				MOV R1,R1	=5539		
				MOV R1,R1	=5540		
				MOV R1,R1	=5541		
				MOV R1,R1	=5542		
				MOV R1,R1	=5543		
				MOV R1,R1	=5544		
				MOV R1,R1	=5545		
				MOV R1,R1	=5546		
				MOV R1,R1	=5547		
				MOV R1,R1	=5548		
				MOV R1,R1	=5549		
				MOV R1,R1	=5550		
				MOV R1,R1	=5551		
				MOV R1,R1	=5552		
				MOV R1,R1	=5553		
				MOV R1,R1	=5554		
				MOV R1,R1	=5555		
				MOV R1,R1	=5556		
				MOV R1,R1	=5557		
				MOV R1,R1	=5558		
				MOV R1,R1	=5559		
				MOV R1,R1	=5560		
				MOV R1,R1	=5561		
				MOV R1,R1	=5562		
				MOV R1,R1	=5563		
				MOV R1,R1	=5564		
				MOV R1,R1	=5565		
				MOV R1,R1	=5566		
				MOV R1,R1	=5567		
				MOV R1,R1	=5568		
				MOV R1,R1	=5569		
				MOV R1,R1	=5570		
				MOV R1,R1	=5571		
				MOV R1,R1	=5572		
				MOV R1,R1	=5573		
				MOV R1,R1	=5574		
				MOV R1,R1	=5575		
				MOV R1,R1	=5576		
				MOV R1,R1	=5577		
				MOV R1,R1	=5578		
				MOV R1,R1	=5579		
				MOV R1,R1	=5580		
				MOV R1,R1	=5581		
				MOV R1,R1	=5582		
				MOV R1,R1	=5583		
				MOV R1,R1	=5584		
				MOV R1,R1	=5585		
				MOV R1,R1	=5586		
				MOV R1,R1	=5587		
				MOV R1,R1	=5588		
				MOV R1,R1	=5589		
				MOV R1,R1	=5590		
				MOV R1,R1	=5591		
				MOV R1,R1	=5592		
				MOV R1,R1	=5593		
				MOV R1,R1	=5594		
				MOV R1,R1	=5595		
				MOV R1,R1	=5596		
				MOV R1,R1	=5597		
				MOV R1,R1	=5598		
				MOV R1,R1	=5599		
				MOV R1,R1	=5600		
				MOV R1,R1	=5601		
				MOV R1,R1	=5602		
				MOV R1,R1	=5603		
				MOV R1,R1	=5604		
				MOV R1,R1	=5605		
				MOV R1,R1	=5606		
				MOV R1,R1	=5607		
				MOV R1,R1	=5608		
				MOV R1,R1	=5609		
				MOV R1,R1	=5610		
				MOV R1,R1	=5611		
				MOV R1,R1	=5612		
				MOV R1,R1	=5613		
				MOV R1,R1	=5614		
				MOV R1,R1	=5615		
				MOV R1,R1	=5616		
				MOV R1,R1	=5617		
				MOV R1,R1	=5618		
				MOV R1,R1	=5619		
				MOV R1,R1	=5620		
				MOV R1,R1	=5621		
				MOV R1,R1	=5622		
				MOV R1,R1	=5623		
				MOV R1,R1	=5624		
				MOV R1,R1	=5625		
				MOV R1,R1	=5626		
				MOV R1,R1	=5627		
				MOV R1,R1	=5628		
				MOV R1,R1	=5629		
				MOV R1,R1	=5630		
				MOV R1,R1	=5631		
				MOV R1,R1	=5632		
				MOV R1,R1	=5633		
				MOV R1,R1	=5634		
				MOV R1,R1	=5635		
				MOV R1,R1	=5636		
				MOV R1,R1	=5637		
				MOV R1,R1	=5638		
				MOV R1,R1	=5639		
				MOV R1,R1	=5640		
				MOV R1,R1	=5641		
				MOV R1,R1	=5642		
				MOV R1,R1	=5643		
				MOV R1,R1	=5644		
				MOV R1,R1	=5645		
				MOV R1,R1	=5646		
				MOV R1,R1	=5647		
				MOV R1,R1	=5648		
				MOV R1,R1	=5649		
				MOV R1,R1	=5650		
				MOV R1,R1	=5651		
				MOV R1,R1	=5652		
				MOV R1,R1	=5653		
				MOV R1,R1	=5654		
				MOV R1,R1	=5655		
				MOV R1,R1	=5656		
				MOV R1,R1	=5657		
				MOV R1,R1	=5658		
				MOV R1,R1	=5659		
				MOV R1,R1	=5660		
				MOV R1,R1	=5661		
				MOV R1,R			

```

=5451 ;
=5452 ;      KEYBOARD AND DISPLAY PROCESSING ROUTINE
=5453 ;      CALLED PERIODICALLY WHEN KBD AND DISPLAY ARE TO BE ALIVE.
074E D5      =5454 TIINT: SEL      RB1
=5455      MMOV      ASAVE, A
074F B93E    =5468+      MOV      R1, #ASAVE
0751 A1      =5469+      MOV      @R1, A
0752 23F0    =5473      MOV      A, #-10H
0754 62      =5474      MOV      T, A      ; RELOAD TIMER INTERVAL
0755 27      =5475      CLR      A
0756 3E      =5476      MOVD     PSEGH, A      ; WRITE BLANK PATTERN TO SEG DRIVERS
0757 3D      =5477      MOVD     PSEGLO, A
0758 FD      =5478      MOV      A, CURDIG
0759 07      =5479      DEC      A
075A 3F      =5480      MOVD     PDIGIT, A      ; ENERGIZE CHARACTER
075B 0C      =5481      MOVD     A, PINPUT      ; LOAD ANY SWITCH CLOSURES
075C AA      =5482      MOV      ROTPAT, A
=5483      ; WRITE NEXT SEGMENT PATTERN
075D FD      =5484      MOV      A, CURDIG
075E 07      =5485      DEC      A
075F 0346    =5486      ADD      A, #SEGMAP      ; ADD CURDIG DISPLACEMENT TO BASE
0761 A8      =5487      MOV      R0, A
0762 F0      =5488      MOV      A, @R0      ; LOAD ACC W/ NEXT SEGMENT PATTERN
0763 3D      =5489      MOVD     PSEGLO, A      ; ENABLE APPROPRIATE SEGMENTS
0764 47      =5490      SWAP     A
0765 3E      =5491      MOVD     PSEGH, A
=5492 ;
=5493 ; *****
=5494 ;      THE NEXT CHARACTER IS NOW BEING DISPLAYED.
=5495 ;      THE KEYBOARD SCAN ROUTINE IS INTEGRATED INTO THE DISPLAY SCAN.
=5496 ;      WITH THE CURRENT ROW ENERGIZED, CHECK IF THERE ARE ANY INPUTS.
=5497 ; *****
=5498 ;
=5499 ;      ROTATE BITS THROUGH THE CY WHILE INCREMENTING KEYLOC.
=5500 ;
0766 B004    =5501      MOV      ROTCNT, #NCOLS      ; SET UP FOR <NCOLS> LOOPS THROUGH 'NXTLOC'
=5502 NXTLOC: MRRC      ROTPAT
0768 FA      =5514+      MOV      A, ROTPAT
0769 67      =5518+      RRC      A
076A AA      =5529+      MOV      ROTPAT, A
=5532      JC      SCAN5      ; ONE BIT IN CY INDICATES KEY NOT DOWN
076B F68B    =5533      MOV      KEYFLG, #1      ; MARK THAT AT LEAST ONE KEY WAS DETECTED
076D BE01    =5534      ; \ IN THE CURRENT SCAN
=5535 ;
=5536 ; *****
=5537 ;      A KEYSTROKE WAS DETECTED FOR THE CURRENT COLUMN. ITS
=5538 ;      POSITION IS IN REGISTER KEYLOC. SEE IF SAME KEY SENSED LAST CYCLE
=5539 ; *****
=5540 ;
=5541      MMOV      A, KEYLOC
076F B93C    =5550+      MOV      R1, #KEYLOC
0771 F1      =5551+      MOV      A, @R1

```

LOC	OBJ	LINE	SOURCE STATEMENT	THINKSTAT2 330402	FILE	LOC	OBJ
0772	2C	=5555	XCH A, LASTKY				
0773	DC	=5556	XRL A, LASTKY				
0774	C67C	=5557	JZ SCAN3				
		=5558 ;					
		=5559 ;	*****				
		=5560 ;	A DIFFERENT KEY WAS READ ON THIS CYCLE THAN ON THE PREVIOUS CYCLE.				
		=5561 ;	SET NREPTS TO THE DEBOUNCE PARAMETER FOR A NEW COUNTDOWN.				
		=5562 ;	*****				
		=5563 ;					
0776	B93D	=5564	MOV R1, #NREPTS				
0778	B106	=5565	MOV @R1, #6				
077A	E488	=5566	JMP SCAN5				
		=5567 ;					
		=5568 ;	*****				
		=5569 ;	SAME KEY WAS DETECTED 35 ON PREVIOUS CYCLE				
		=5570 ;	LOOK AT NREPTS: IF ALREADY ZERO, DO NOTHING.				
		=5571 ;	ELSE DECREMENT NREPTS.				
		=5572 ;	IF THIS RESULTS IN ZERO, MOVE LASTKY INTO KBDGUF.				
		=5573 ;	*****				
		=5574 ;					
		=5575	SCAN3: MMOV A, NREPTS				
077C	B93D	=5584+	MOV R1, #NREPTS				
077E	F1	=5585+	MOV A, @R1				
077F	C68B	=5589	JZ SCAN5 ; IF ALREADY ZERO				
0781	07	=5590	DEC A ; INDICATE ONE MORE SUCCESSION KEY DETECTION				
		=5591	MMOV NREPTS, A				
0782	B93D	=5604+	MOV R1, #NREPTS				
0784	A1	=5605+	MOV @R1, A				
0785	968B	=5609	JNZ SCAN5 ; IF DECREMENT DOES NOT RESULT IN ZERO				
		=5610	MMOV KBDGUF, LASTKY ; TO MARK NEW KEY CLOSURE				
0787	FC	=5633+	MOV A, LASTKY				
0788	B93B	=5639+	MOV R1, #KBDGUF				
078A	A1	=5640+	MOV @R1, A				
		=5643 ;					
078B	B93C	=5644	SCAN5: MOV R1, #KEYLOC				
078D	11	=5645	INC @R1				
078E	ED68	=5646	DJNZ ROTCNT, NXTLOC				
0790	ED68	=5647	DJNZ CURDIG, TIRET1				
0792	B008	=5648	MOV CURDIG, #CHARNO				
		=5649 ;					
		=5650 ;	*****				
		=5651 ;	THE FOLLOWING CODE SEGMENT IS USED BY THE KEYBOARD SCANNING ROUTINE.				
		=5652 ;	IT IS EXECUTED ONLY AFTER A REFRESH SEQUENCE IS COMPLETED				
		=5653 ;	*****				
		=5654 ;					
		=5655	MMOV KEYLOC, ZERO				
0794	B93C	=5666+	MOV R1, #KEYLOC				
0796	B100	=5667+	MOV @R1, #ZERO				
0798	FE	=5671	MOV A, KEYFLG				
0799	969D	=5672	JNZ SCAN8 ; JUMP IF ANY KEYS WERE DETECTED				
		=5673	MMOV LASTKY, NEG1 ; CHANGE <LASTKY> WHEN NO KEYS ARE DOWN				
079B	BCFF	=5678+	MOV LASTKY, #NEG1				
079D	BE00	=5682	SCAN8: MOV KEYFLG, #0				
		=5683 ;					
		=5684 ;	*****				

LOC	OBJ	LINE	SOURCE STATEMENT	STATEMENT	LINE	LOC
		=5685 ;		MOV A, LASTKEY	=5685	0770 30
		=5686 ;	KBD/DISP RETURN CODE- RESTORES SYSTEM STATUS.	MOV A, LASTKEY	=5686	0770 30
		=5687	MMOV A, RDELAY	MOV A, LASTKEY	=5687	0770 30
079F B93F		=5696+	MOV R1, #RDELAY		=5696	
07A1 F1		=5697+	MOV A, @R1		=5697	
07A2 C6A8		=5701	JZ TIRET1		=5701	
07A4 07		=5702	DEC A		=5702	
		=5703	MMOV RDELAY, A		=5703	
07A5 B93F		=5716+	MOV R1, #RDELAY		=5716	
07A7 A1		=5717+	MOV @R1, A		=5717	
		=5721	TIRET1: MMOV A, ASAVE		=5721	
07A8 B93E		=5730+	MOV R1, #ASAVE		=5730	
07AA F1		=5731+	MOV A, @R1		=5731	
07AB 93		=5735	RETR		=5735	
		=5736 ;			=5736	
		=5737 ;			=5737	
		=5738 ;	TOFPOL TIMER OVERFLOW POLLING SUBROUTINE.		=5738	
		=5739 ;	CALLLED REPEATEDLY FROM WHEREVER KBD/DISP MUST BE ALIVE.		=5739	
		=5740 ;	MONITORS THE TIMER OVERFLOW FLAG (TOF) AND CALLS SERVICE		=5740	
		=5741 ;	ROUTINE WHEN APPROPRIATE.		=5741	
07AC 164E		=5742	TOFPOL: JTF TIINT		=5742	
07AE 83		=5743	RET		=5743	
		=5744	SIZECHK		=5744	
0061		=5747+	SIZE SET 97		=5747	
		=5748+			=5748	
		=5749+;	*****		=5749	
		=5750	\$EJECT		=5750	

LOC	OBJ	LINE	SOURCE STATEMENT	THEMSTATE2	LINE	LOC	OBJ
		=5759	CODEBLK 17	#130007K	VON	01020	0000
06C2		=5709+	ORG 1730	T2207K	JUL	00020	0000
		=5793 ;		JATXCH	MOV	00021	0000
		=5794 ;	KBDIN: KEYBOARD INPUT SUBROUTINE.	JATXCH	MOV	00022	0000
		=5795 ;	RETURNS ONLY AFTER A NEW KEYSTROKE HAS BEEN DETECTED AND DEBOUNCED.			00023	0000
		=5796 ;	VALUE OF KEY POSITION IN SWITCH MATRIX IS			00024	0000
		=5797 ;	RETURNED IN THE ACCUMULATOR.			00025	0000
		=5798 ;	DISPLAY CHARACTER NOW ON BLANKED BEFORE RETURNING.			00026	0000
06C2 DF03		=5799 KBDIN:	MOV XPCODE, #3			00027	0000
06C4 74D1		=5800	CALL XPTST			00028	0000
06C6 F4AC		=5801 KBDI1:	CALL TOPPOL			00029	0000
		=5802	MMOV A, KBDUF			00030	0000
06C0 B93B		=5811+	MOV R1, #KBDUF			00031	0000
06CA F1		=5812+	MOV A, @R1			00032	0000
06CB F206		=5816	JB7 KBDI1			00033	0000
06CD 27		=5817	CLR A			00034	0000
06CE 3E		=5818	MOVD PSEGL1, A			00035	0000
06CF 3D		=5819	MOVD PSEGL0, A			00036	0000
06D0 37		=5820	CPLM #0, A			00037	0000
06D1 21		=5821	XCH A, @R1			00038	0000
06D2 83		=5822	RET			00039	0000
		=5823	SIZECHK			00040	0000
0011		=5826+	SIZE SET 17			00041	0000
		=5827+				00042	0000
		=5828+;	*****			00043	0000
		=5837 ;				00044	0000
		=5838	CODEBLK 15			00045	0000
05F1		=5863+	ORG 010 1521			00046	0000
		=5867 ;	CLEAR WRITES BLANK CHARACTERS INTO ALL DISPLAY REGISTERS.			00047	0000
		=5868 ;	RETURNS WITH NEXTPL SET TO LEFTMOST CHARACTER POSITION			00048	0000
		=5869 ;	DOES NOT AFFECT ACC OR CY.			00049	0000
05F1 B046		=5870 CLEAR:	MOV R0, #SEGMAP			00050	0000
05F3 B908		=5871	MOV R1, #CHARNO			00051	0000
05F5 B000		=5872 DBLANK:	MOV @R0, #0			00052	0000
05F7 18		=5873	INC R0			00053	0000
05F8 E9F5		=5874	DJNZ @R1, DBLANK			00054	0000
		=5875	MMOV @NEXTPL, CHARNO			00055	0000
05FA B93A		=5886+	MOV @R1, #NEXTPL			00056	0000
05FC B108		=5887+	MOV @R1, #CHARNO			00057	0000
05FE 83		=5891	RET			00058	0000
		=5892	SIZECHK			00059	0000
000E		=5895+ SIZE SET 14				00060	0000
		=5896+;				00061	0000
		=5897+;	*****			00062	0000
		=5906 ;				00063	0000
		=5907	CODEBLK 44			00064	0000
06D3		=5937+	ORG 1747			00065	0000
		=5941 ;	DSPACC DISPLAY VALUE OF LOW NIBBLE OF ACC			00066	0000
06D3 530F		=5942 DSPACC:	ANL A, #0FH			00067	0000
06D5 03EF		=5943	ADD @A, #DGPATS			00068	0000
06D7 A3		=5944	MOV @A, @R1			00069	0000
		=5945 ;	WDISP WRITES BIT PATTERN NOW IN ACC INTO NEXT CHARACTER POSITION			00070	0000
		=5946 ;	OF THE DISPLAY (NEXTPL). INCREMENTS NEXTPL			00071	0000
		=5947 ;	RESULTS IN DISPLAY BEING FILLED LEFT TO RIGHT, THEN RESTARTING			00072	0000
06D8 AE		=5948 WDISP:	MOV DSPTMP, A			00073	0000

	=5951	MMOV	A, NEXTPL		
06DD B93A	=5960+	MOV	R1, #NEXTPL		
06DF F1	=5961+	MOV	R1, @R1		
06E0 0345	=5965	ADD	A, #SEGMAP-1		
06E2 A9	=5966	MOV	R1, A		
06E3 FE	=5967	MOV	A, D5PTMP		
06E4 A1	=5968	MOV	@R1, A		
	=5969	MDJNZ	NEXTPL, WDISP1		
06E5 B93A	=5974+	MOV	R1, #NEXTPL		
06E7 F1	=5975+	MOV	A, @R1		
06E8 07	=5979+	DEC	A		
06E9 A1	=5984+	MOV	@R1, A		
06EA 96EE	=5988+	JNZ	WDISP1		
06EC B108	=5990	MOV	@R1, #CHARNO		
06EE 83	=5991	WDISP1: RET			
	=5992 ;				
	=5993 ;	DGPATS IS THE BASE FOR THE TABLE OF SEGMENT PATTERNS FOR HEX DIGITS.			
	=5994 ;	HERE THE FULL HEX SET (0-F) IS INCLUDED.			
	=5995 ;				
00EF	=5996	DGPATS EQU	\$ AND 0FFH		
	=5997 ;				
	=5998 ;	FORMAT IS	PGFEDCBA		
	=5999 ;				
06EF 3F	=6000	DB	00111111B		
06F0 06	=6001	DB	00000110B		
06F1 58	=6002	DB	01011011B		
06F2 4F	=6003	DB	01001111B		
06F3 66	=6004	DB	01100110B		
06F4 6D	=6005	DB	01101101B		
06F5 7D	=6006	DB	01111101B		
06F6 07	=6007	DB	00000111B		
06F7 7F	=6008	DB	01111111B		
06F8 67	=6009	DB	01100111B		
06F9 77	=6010	DB	01110111B		
06FA 7C	=6011	DB	01111100B		
06FB 39	=6012	DB	00111001B		
06FC 5E	=6013	DB	01011110B		
06FD 79	=6014	DB	01111001B		
06FE 71	=6015	DB	01110001B		
	=6016	SIZECHK			
002C	=6019+	SIZE SET	44		
	=6020+;				
	=6021+;	*****			
	=6030 ;				
	=6031	CODEBLK 12			
04F2	=6051+	ORG	1266		
	=6055 ;	DELAY	SUBROUTINE WAITS FOR THE NUMBER OF COMPLETE		
	=6056 ;		DISPLAY SCANS CORRESPONDING TO THE ACC CONTENTS.		
	=6057 ;		USED WITH CRUDE HUMAN INTERFACES- AS WHEN OPERATOR SHOULD SEE		
	=6058 ;		SOME DISPLAY CHANGE WHILE IT IS CHANGING.		
	=6059	DELAY: MMOV	RDELAY, A		
04F2 B93F	=6072+	MOV	R1, #RDELAY		
04F4 A1	=6073+	MOV	@R1, A		

LOC	OBJ	LINE	SOURCE STATEMENT	OBJECT STATEMENT	LINE	LOC	OBJ
04F5	F4AC	=6077	DELAY1: CALL 7OFFOL	CALL 7OFFOL	6077		
		=6078	MMOV A, RDELAY	MMOV A, RDELAY	6078		
04F7	B93F	=6087+	MOV R1, #RDELAY	MOV R1, #RDELAY	6087		
04F9	F1	=6088+	MOV A, @R1	MOV A, @R1	6088		
04FA	96F5	=6092	JNZ DELAY1	JNZ DELAY1	6092		
04FC	83	=6093	RET	RET	6093		
		=6094	SIZECHK	SIZECHK	6094		
0008		=6097+	SIZE SET 11	SIZE SET 11	6097		
		=6098;			6098		
		=6099+;	*****	*****	6099		
		=6100 ;			6100		
		=6109	CODEBLK 8	CODEBLK 8	6109		
07AF		=6144+	ORG 1967	ORG 1967	6144		
		=6148 ;	KBDPOL POLL STATUS OF KEYBOARD INPUT ROUTINE.		6148		
		=6149 ;	RETURN WITH ACC BIT 7 = 0 IF KEYBOARD INPUT HAS BEEN RECEIVED.		6149		
07AF	BF05	=6150	KBDPOL: MOV XPCODE, #5	MOV XPCODE, #5	6150		
07B1	74D1	=6151	CALL XPTST	CALL XPTST	6151		
		=6152	MMOV A, KBDBUF	MMOV A, KBDBUF	6152		
07B3	B93B	=6161+	MOV R1, #KBDBUF	MOV R1, #KBDBUF	6161		
07B5	F1	=6162+	MOV A, @R1	MOV A, @R1	6162		
07B6	83	=6166	RET	RET	6166		
		=6167	SIZECHK	SIZECHK	6167		
0008		=6170+	SIZE SET 8	SIZE SET 8	6170		
		=6171+;			6171		
		=6172+;	*****	*****	6172		
		=6181	\$EJECT	\$EJECT	6181		

LOC	OBJ	LINE	SOURCE STATEMENT	THIRDTAT2 33RU02	LINE	LOC
		6182 \$	INCLUDE(:F0:LINK.MOD)			
		=6183	CODEBLK 15			
07B7		=6218+	ORG 1975			
		=6222 ;EPFET	FETCH DATA BYTE FROM EP INTERNAL RAM ADDRESSED BY SMALO.			
		=6223 EPFET:	MMOV A, SMALO			
07B7 B930		=6232+	MOV R1, #SMALO			
07B9 F1		=6233+	MOV A, @R1			
07BA F4D0		=6237	CALL EPPASS			
07BC 2380		=6238	MOV A, #10000000B			
07BE F4D0		=6239	CALL EPPASS			
07C0 F4D0		=6240	CALL EPPASS			
07C2 83		=6241	RET			
		=6242	SIZECHK			
000C		=6245+ SIZE SET 12				
		=6246+;				
		=6247+; *****				
		=6256 ;				
		=6257	CODEBLK 15			
07C3		=6292+	ORG 1987			
		=6296 ;EPSTOR	STORE DATA IN LDATA IN EP INTERNAL RAM AT <SMALO>			
07C3 FA		=6297 EPSTOR:	MOV A, LDATA			
07C4 F4D0		=6298	CALL EPPASS			
		=6299	MMOV A, SMALO			
07C6 B930		=6308+	MOV R1, #SMALO			
07C8 F1		=6309+	MOV A, @R1			
07C9 537F		=6313	ANL A, #01111111B			
07CB F4D0		=6314	CALL EPPASS			
07CD F4D0		=6315	CALL EPPASS			
07CF 83		=6316	RET			
		=6317	SIZECHK			
000D		=6320+ SIZE SET 13				
		=6321+;				
		=6322+; *****				
		=6331 \$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	THOMAS2 30/002	3M1	100 301
		=6332 ;	THE FOLLOWING UTILITIES INVOLVE INTERCHANGES BETWEEN THE MP AND EP.			
		=6333 ;				
		=6334	CODEBLK 11			
07D0		=6369+	ORG 2000			
		=6373 ;	EPPASS PASSES A SINGLE PARAMETER BYTE TO THE EP THROUGH THE LINK.			
		=6374 ;	WRITE THE CONTENTS OF THE ACC TO THE LINK;			
		=6375 ;	RELEASE THE EP;			
		=6376 ;	READ THE LINK INTO THE ACC;			
		=6377 ;	RETURN.			
07D0 8A30		=6378 EPPASS:	ORL P2, #00110000B ;	ENABLE LINK WRITES.		
07D2 91		=6379	MOVX R1, A ;	WRITE ACC TO LINK.		
07D3 99FE		=6380	ANL P1, #NOT ENBRAM ;	DISABLE BREAKPOINTS.		
07D5 8902		=6381	ORL P1, #ENBLNK ;	SET TO BREAK ON LINK REFERENCE.		
07D7 F4DB		=6382	CALL EPSTEP			
07D9 81		=6383	MOVX A, R1			
07DA 83		=6384	RET			
		=6385	SIZECHK			
000B		=6388+ SIZE SET 11				
		=6389+;				
		=6390+;	*****			
		=6399 ;				
		=6400	CODEBLK 25			
07DB		=6435+	ORG 2011			
		=6439 ;	EPSTEP RELEASES EP TO RUN IN PRESENT MODE UNTIL AN ANTICIPATED			
		=6440 ;	HARDWARE BREAK OCCURS.			
		=6441 ;	XXXX (DUE TO SINGLE STEPPING, LINK OPCODE FETCH, OR LINK DATA FETCH)			
		=6442 ;	MUST OCCUR WITHIN A FINITE NUMBER OF CYCLES (<40 MP CYCLES)			
		=6443 ;	OR WATCHDOG TIMER WILL ASSUME A COMMUNICATIONS ERROR			
		=6444 ;	BETWEEN THE MP AND EP.			
07DB F4F4		=6445 EPSTEP:	CALL EPREL			
07DD B90A		=6446	MOV R1, #10			
07DF 86F1		=6447 EPSTE1:	JNI EPSTE2			
07E1 E9DF		=6448	DJNZ R1, EPSTE1			
07E3 8910		=6449	ORL P1, #EPRSET			
07E5 744F		=6450	CALL EPBRK			
07E7 B8BB		=6451	MOV R0, #LOW(OVLBAS+OVSZ)			
07E9 746A		=6452	CALL OVLORD			
07EB 99EF		=6453	ANL P1, #NOT EPRSET			
07ED BA0E		=6454	MOV LDAT, #0EH			
07EF 249A		=6455	JMP PERROR			
07F1 744F		=6456 EPSTE2:	CALL EPBRK			
07F3 83		=6457	RET			
		=6458	SIZECHK			
0019		=6461+ SIZE SET 25				
		=6462+;				
		=6463+;	*****			
		=6472 ;				
		=6473 ;				
		=6474 ;	EJECT			

	=6514 ;	EPREL	RELEASES EP TO RUN IN PRESENT MODE.	
	=6515 ;		SEQUENCE IS AS FOLLOWS:	
	=6516 ;		PUT MEMORY ARRAY IN EP MODE;	
	=6517 ;		RAISE /SSTEP; (ONLY AT THE END OF THE EP)	
	=6518 ;		RETURN.	
07F4 99F7	=6519 EPREL:	ANL	P1,#NOT CLRBF	;CLEAR BREAK F/F
07F6 8908	=6520	ORL	P1,#CLRBF	;RE-ENABLE BREAK F/F
07F8 9ABF	=6521	ANL	P2,#NOT 01000000B	;ENABLE EP CONTROL OF MEM ARRAY
07FA 8904	=6522	ORL	P1,#00000100B	;FREE EP TO RUN UNTIL BREAK.
07FC 83	=6523	RET		
	=6524	SIZECHK		
0009	=6527+ SIZE	SET	9	
	=6528+;			
	=6529+;			
	=6530 ;			
	=6539 ;			
	=6540	CODEBLK	11	
034F	=6580+	ORG	847	
	=6584 ;	EPBRK	REGAIN CONTROL OF MEMORY ARRAY FROM EP.	
	=6585 ;		DROP /SSTEP;	
	=6586 ;		WAIT 30 USECS.;	
	=6587 ;		PUT MEMORY ARRAY IN MP MODE;	
	=6588 ;		RETURN.	
034F 99FB	=6589 EPBRK:	ANL	P1,#NOT 00000100B	;FREEZE EMULATION PROCESSOR.
0351 8920	=6590	ORL	P1,#MODOUT	;SIGNAL EP IS NOT RUNNING USER CODE.
0353 8905	=6591	MOV	R1,#5	
0355 E955	=6592	DJNZ	R1,\$;DELAY FOR EP TO FINISH INSTRUCTION.
0357 8A40	=6593	ORL	P2,#01000000B	;SEIZE CONTROL OF MEM ARRAY.
0359 83	=6594	RET		
	=6595	SIZECHK		
000B	=6598+ SIZE	SET	11	
	=6599+;			
	=6600+;			
	=6609 ;			
	=6610 ;			
	=6611	CODEBLK	16	
035A	=6651+	ORG	858	
	=6655 ;	OVSMP	OVERLAY SWMP.	
	=6656 ;		SWAPS BLOCK OF DATABYTES (USER'S PROGRAM) BETWEEN MP RAM & EP PM.	
035A B865	=6657 OVSMP:	MOV	R0,#OVBUF+OVSIZE	
035C B917	=6658	MOV	R1,#OVSIZE	
035E 2340	=6659	MOV	R,#01000000B	
0360 3A	=6660	OUTL	P2,R	
0361 C8	=6661 OVSMP1:	DEC	R0	
0362 C9	=6662	DEC	R1	
0363 81	=6663	MOVX	R,#R1	
0364 20	=6664	XCH	R,#R0	
0365 91	=6665	MOVX	R1,R	
0366 F9	=6666	MOV	R,R1	
0367 9661	=6667	JNZ	OVSMP1	
0369 83	=6668	RET		
	=6669	SIZECHK		
0010	=6672+ SIZE	SET	16	

LOC	OBJ	LINE	SOURCE STATEMENT	THGHTATZ 33002	INTJ	LOC 001
		=6673+;				
		=6674+; *****				
		=6683 ;				
		=6684 ; CODEBLK 14				
036A		=6724+ ORG 874				
		=6728 ; OVLORD OVERLAY LOAD				
		=6729 ; MOVES BLOCK OF DATABYTES (ASSEMBLED SOURCE) FROM PG3 TO EP PM				
		=6730 ; TOP OF DATA BLOCK LOADED AND BLOCK LENGTH DETERMINED BY R0 AND R1				
036A B917		=6731 OVLORD: MOV R1, #OVSIZE				
036C 2340		=6732 MOV A, #01000000B				
036E 3A		=6733 OUTL P2, A				
036F C8		=6734 MML01: DEC R0				
0370 C9		=6735 DEC R1				
0371 F8		=6736 MOV A, R0				
0372 E3		=6737 MOV P3, A				
0373 91		=6738 MOVX @R1, A				
0374 F9		=6739 MOV A, R1				
0375 966F		=6740 JNZ MML01				
0377 83		=6741 RET				
		=6742 SIZECHK				
000E		=6745+ SIZE SET 14				
		=6746+;				
		=6747+; *****				
		=6756 \$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	THIRDTATZ-303002	TIME	LOC	OBJ
		=6757 ;					
		=6758 ;	=====				
		=6759 ;					
		=6760 ;	THE REST OF THIS MODULE CONTAINS THE MINI-MONITORS WHICH OVERLAY				
		=6761 ;	THE EMULATION PROCESSOR PROGRAM RAM TO GIVE THE				
		=6762 ;	MASTER PROCESSOR ACCESS TO INTERNAL REGISTERS AND RAM OF THE EP.				
		=6763 ;	=====				
		=6764 ;	=====				
		=6765 ;					
		=6766	DATABLK 22				
0378		=6771+	ORG 000				
		=6775 ;					
		=6776 ;	OVERLAY TO BREAK EP EXECUTION AND JUMP TO LOCATION 009H.				
		=6777 ;	LOCATION 009H REACHED WITH TOP-OF-STACK = RETURN ADDRESS+2				
		=6778 ;	DUE TO FORCED "CALL" DURING WHICH PC WAS INCREMENTED.				
		=6779 ;	LOCS 003H & 007H CALL 009H TO SIMULATE SAME CONDITION				
		=6780 ;	IF BREAK OCCURS DURING INTERRUPT CYCLE.				
		=6781 ;	SOURCE CODE FOR MINI-MONITOR OVERLAYED OVER LOW ORDER PROGRAM RAM.				
		=6782 ;					
0378		=6783 OV0BAS	EQU \$				
0378		=6784 ORG	OV0BAS				
0378 1409		=6785	CALL 009H				
037A 00		=6786	NOP				
		=6787 ;					
037B		=6788 ORG	OV0BAS+003H				
037B 1409		=6789	CALL 009H				
037D 00		=6790	NOP				
037E 00		=6791	NOP				
		=6792 ;					
037F		=6793 ORG	OV0BAS+007H				
037F 1409		=6794	CALL 009H				
0381 00		=6795	NOP				
0382 00		=6796	NOP				
0383 00		=6797	NOP				
0384 00		=6798	NOP				
0385 00		=6799	NOP				
0386 00		=6800	NOP				
0387 00		=6801	NOP				
0388 00		=6802	NOP				
0389 00		=6803	NOP				
038A 00		=6804	NOP				
038B 00		=6805	NOP				
		=6806 ;					
038C		=6807 ORG	OV0BAS+014H				
038C 0409		=6808	JMP 009H				
		=6809 ;					
		=6810	SIZECHK				
0016		=6813+ SIZE SET 22					
		=6814+;					
		=6815+; *****					
		=6824 \$EJECT					

LOC	OBJ	LINE	SOURCE STATEMENT	ASSEMBLY STATEMENT	LINE	LOC	OBJ
		=6825	DATABLK 22	SS DATABLK	6825		
038E		=6830+	ORG 910	ORG 910	6830		
		=6834 ;			6834		
		=6835 ; OV3-	OVERLAY TO SAVE STATUS DATA AFTER BREAK.	TO OV3 VALMEMO	6835		
		=6836 ;	ACC, TIMER/COUNTER, PSW (WITH F1), & RAM LOC 0 PASSED SEQUENTIALLY		6836		
		=6837 ;	TO MP.		6837		
		=6838 ;	SOURCE CODE FOR MINI-MONITOR OVERLAYED OVER LOW ORDER PROGRAM RAM.		6838		
		=6839 ;			6839		
038E		=6840 OV3BAS	EQU \$	OV3BAS	6840		
038E		=6841 ORG	OV3BAS	ORG	6841		
038E 0400		=6842	JMP 000H	JMP	6842		
0390 00		=6843	NOP	NOP	6843		
		=6844 ;			6844		
0391		=6845 ORG	OV3BAS+003H	ORG	6845		
0391 83		=6846	RET	RET	6846		
0392 00		=6847	NOP	NOP	6847		
0393 00		=6848	NOP	NOP	6848		
0394 00		=6849	NOP	NOP	6849		
		=6850 ;			6850		
0395		=6851 ORG	OV3BAS+007H	ORG	6851		
0395 83		=6852	RET	RET	6852		
0396 00		=6853	NOP	NOP	6853		
		=6854 ;			6854		
0397		=6855 ORG	OV3BAS+009H	ORG	6855		
0397 90		=6856	MOVX @R0, A	MOVX @R0, A	6856		
0398 42		=6857	MOV A, T	MOV A, T	6857		
0399 90		=6858	MOVX @R0, A	MOVX @R0, A	6858		
039A C7		=6859	MOV A, PSW	MOV A, PSW	6859		
039B 7611		=6860	JF1 OV3B1	JF1 OV3B1	6860		
039D 53F7		=6861	ANL A, #11110111B	ANL A, #11110111B	6861		
0311		=6862 OV3B1	EQU \$- (LOW OV3BAS)	OV3B1	6862		
039F 90		=6863	MOVX @R0, A	MOVX @R0, A	6863		
03A0 C5		=6864	SEL R00	SEL R00	6864		
03A1 F0		=6865	MOV A, R0	MOV A, R0	6865		
03A2 0409		=6866	JMP 009H	JMP 009H	6866		
		=6867 ;			6867		
		=6868	SIZECHK		6868		
0016		=6871+ SIZE	SET 22	SET 22	6871		
		=6872+;			6872		
		=6873+; *****			6873		
		=6882 \$EJECT			6882		

```

=6892 ;
=6893 ; OV1- OVERLAY 1 TO GIVE MP ACCESS TO EP RAM LOCS. 01H-7FH.
=6894 ; SOURCE CODE FOR MINI-MONITOR OVERLAYED OVER LOW ORDER PROGRAM RAM.
=6895 ;
03A4 =6896 OV1BAS EQU $
=6897 ;
03A4 040A =6898 JMP OV1B1
03A6 00 =6899 NOP
=6900 ;
03A7 =6901 ORG OV1BAS+003H
03A7 83 =6902 RET
03A8 00 =6903 NOP
03A9 00 =6904 NOP
03AA 00 =6905 NOP
=6906 ;
03AB =6907 ORG OV1BAS+007H
03AB 83 =6908 RET
03AC 00 =6909 NOP
=6910 ;
03AD =6911 ORG OV1BAS+009H
03AD 90 =6912 MOVX @R0, A
=6913 ;
000A =6914 OV1B1 EQU $-OV1BAS
=6915 ;
03AE 80 =6916 MOVX A, @R0
03AF A8 =6917 MOV R0, A
03B0 80 =6918 MOVX A, @R0
03B1 F213 =6919 JB7 OV1B2
03B3 28 =6920 XCH A, R0
03B4 A0 =6921 MOV @R0, A
03B5 0409 =6922 JMP 009H
=6923 ;
0313 =6924 OV1B2 EQU $-LOW OV1BAS
=6925 ;
03B7 F0 =6926 MOV A, @R0
03B8 0409 =6927 JMP 009H
=6928 ;
=6929 SIZECHK
0016 =6932+ SIZE SET 22
=6933+;
=6934+; *****
=6943 $EJECT

```


LOC	OBJ	LINE	SOURCE STATEMENT	OBJECT STATEMENT	LOC	OBJ
		=6944	DATABLK 23			
030A		=6949+	ORG 954			
		=6953 ;				
		=6954 ; OV2-	OVERLAY TO RESTORE EP STATUS SAVED ON BREAK AND RESUME USER'S PROGRAM.			
		=6955 ;	SOURCE CODE FOR MINI-MONITOR OVERLAYED OVER LOW ORDER PROGRAM RAM.			
		=6956 ;				
030A		=6957 OV2BAS	EQU \$			
030A		=6958 ORG	OV2BAS			
030A 0400		=6959	JMP 000H			
030C 00		=6960	NOP			
		=6961 ;				
030D		=6962 ORG	OV2BAS+003H			
030D 83		=6963	RET			
030E 00		=6964	NOP			
030F 00		=6965	NOP			
0300 00		=6966	NOP			
		=6967 ;				
03C1		=6968 ORG	OV2BAS+007H			
03C1 83		=6969	RET			
03C2 00		=6970	NOP			
		=6971 ;				
03C3		=6972 ORG	OV2BAS+009H			
03C3 90		=6973	MOVX @R0, A			
		=6974 ;				
03C4 80		=6975	MOVX A, @R0			
03C5 A8		=6976	MOV R0, A			
03C6 80		=6977	MOVX A, @R0			
03C7 D7		=6978	MOV PSW, A			
03C8 A5		=6979	CLR F1			
03C9 B5		=6980	CPL F1			
03CA 7213		=6981	JB3 OV2B1			
03CC A5		=6982	CLR F1			
		=6983 ;				
0313		=6984 OV2B1	EQU \$-LOW OV2BAS			
		=6985 ;				
03CD 80		=6986	MOVX A, @R0			
03CE 62		=6987	MOV T, A			
03CF 80		=6988	MOVX A, @R0			
03D0 93		=6989	RETR			
		=6990	SIZECHK			
0017		=6993+ SIZE	SET 23			
		=6994+;				
		=6995+; *****				
		=7004 \$EJECT				

3-168

LOC	OBJ	LINE	SOURCE STATEMENT	DISASSEMBLY	HEX	DEC
		7145 ;	*****	TEST		
		7146 ;		TEST		
		7147 ;	FILL ALL UNUSED MEMORY LOCATIONS WITH NOP OPCODES	TEST		00 7700
		7148 ;		TEST		
		7149 ;	*****	TEST		7700
		7150 ;	(20000 - 0000) TEST	TEST		
		7151 \$GEN		TEST		
		7158 ;		TEST		
01FD		7160	ORG ORGPG1	TEST		00 7700
		7161	REPT (200H - ORGPG1)	TEST		
		7162	DB 0	TEST		00 7700
		7163	ENDM	TEST		
01FD 00		7164+	DB 0	TEST		
01FE 00		7165+	DB 0	TEST		
01FF 00		7166+	DB 0	TEST		00 0700
		7168 ;		TEST		00 0700
		7175 ;		TEST		00 0700
03E9		7177	ORG ORGPG3	TEST		
		7178	REPT (400H - ORGPG3)	TEST		
		7179	DB 0	TEST		
		7180	ENDM	TEST		
03E9 00		7181+	DB 0	TEST		
03EA 00		7182+	DB 0	TEST		
03EB 00		7183+	DB 0	TEST		
03EC 00		7184+	DB 0	TEST		
03ED 00		7185+	DB 0	TEST		
03EE 00		7186+	DB 0	TEST		
03EF 00		7187+	DB 0	TEST		
03F0 00		7188+	DB 0	TEST		
03F1 00		7189+	DB 0	TEST		
03F2 00		7190+	DB 0	TEST		
03F3 00		7191+	DB 0	TEST		
03F4 00		7192+	DB 0	TEST		
03F5 00		7193+	DB 0	TEST		
03F6 00		7194+	DB 0	TEST		
03F7 00		7195+	DB 0	TEST		
03F8 00		7196+	DB 0	TEST		
03F9 00		7197+	DB 0	TEST		
03FA 00		7198+	DB 0	TEST		
03FB 00		7199+	DB 0	TEST		
03FC 00		7200+	DB 0	TEST		
03FD 00		7201+	DB 0	TEST		
03FE 00		7202+	DB 0	TEST		
03FF 00		7203+	DB 0	TEST		
		7205 ;		TEST		
04FD		7207	ORG ORGPG4	TEST		
		7208	REPT (500H - ORGPG4)	TEST		
		7209	DB 0	TEST		
		7210	ENDM	TEST		
04FD 00		7211+	DB 0	TEST		
04FE 00		7212+	DB 0	TEST		
04FF 00		7213+	DB 0	TEST		
		7215 ;		TEST		
05FF		7217	ORG ORGPG5	TEST		
		7218	REPT (600H - ORGPG5)	TEST		

All trademarks copyrighted © Intel Corporation 1976.

All trademarks copyrighted © Intel Corporation 1978.

All trademarks copyrighted © Intel Corporation 1976.

All trademarks copyrighted © Intel Corporation 1978

PSÉGLO 000C	RDELAY 003F	RECDON 02CC	RECTVP 0042	KEGC 0044	REORG 0005	RERROR 0198	RINI 0011
ROTCNT 0003	ROTPAT 0002	RSOURC 0012	SCAN3 077C	SCAN5 078B	SCAN8 079D	SEGMAP 0046	SING 001A
SIZE 000D	SIZECH 0011	SMWHT 0031	SMALO 0030	STRCOM 001D	STRGOC 002C	STRMEM 0026	STRTMP 0040
STRUTL 0019	STSAVE 0500	TCRLFO 01D2	TIINT 074E	TIRET1 07A8	TOPOL 07AC	TIVOUT 0040	TYPE 0037
UPDAD1 017C	UPDADR 0178	VERSNO 0029	WBRK 0016	WDISP 06D8	WDISP1 06EE	XPCODE 0007	XPIEST 03D1
ZERO 0000							

ASSEMBLY COMPLETE, NO ERRORS

0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991	0992	0993	0994	0995	0996	0997	0998	0999
------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

ISIS-II ASSEMBLER SYMBOL CROSS REFERENCE, V2.1																
NAME	0012	0050	0052	0070	0072	0070	0072	0070	0072	0070	0072	0070	0072	0070	0072	0070
?A	105#	1614	1629	1637	1650	1658	1721	1787	1806	1818	1977	1985	1999	2177	2388	2444
?B	2546	2586	2719	2788	2796	2843	2857	2865	2898	2910	2928	2943	2958	2973	3007	3068
?C	3096	3207	3215	3226	3234	3245	3253	3264	3272	3288	3302	3310	3323	3331	3350	3358
	3434	3442	3453	3461	3472	3480	3491	3499	3515	3562	3583	3637	3958	3966	3986	4001
	4016	4070	4393	4401	4592	4764	4788	4803	4819	4841	4859	4875	4962	4986	5001	5017
	5042	5095	5167	5180	5196	5348	5360	5374	5386	5456	5464	5546	5580	5592	5600	5692
	5704	5712	5726	5807	5956	6060	6068	6083	6157	6228	6304					
?ASAVE	1235#	5460	5466	5722	5728											
?B	1280#	4413	4413	4413	4419	4459	4469	4569	4579							
?B0PNT	117#	746	754#	763	771#	780	788#	797	805#	814	822#	831	839#			
?B0R2	110#	754														
?B0R3	111#	771														
?B0R4	112#	788														
?B0R5	113#	805														
?B0R6	114#	822														
?B0R7	115#	839														
?B1PNT	126#	859	867#	880	888#	901	909#	922	930#	943	951#					
?B1R2	119#	867														
?B1R3	120#	888														
?B1R4	121#	909														
?B1R5	122#	930														
?B1R6	123#	951														
?B1R7	124#															
?BCODE	1163#	1561	1561	1561	1567	1610	1616	2390								
?BINOP	415#	1817	2387	2439	2909	3632	5179	5359	5385							
?BIT50	4217#	4411														
?BUFCN	1262#	3438	3444	3511	3517	3532	3542	3962	3968	3982	3988	4044	4054			
?BUFILE	649#															
?CHARN	585#	5876														
?CHKSU	791#	3426	3426	3558	3564	3571	3571	3858	3858	3858	4066	4072	4079	4079		
?CONST	104#	585	586	590	594	601	602	606	610	617	618	622	626	633	634	638
	642	649	650	654	658	671	672	676	680	686	687	691	695	701	702	706
	710	716	717	721	725	4217	4218	4222	4226							
?CURDI	912#															
?DEBNC	617#															
?DSPTI	1037#	3092	3098													
?DSPTM	808#															
?ENRHI	1136#	5388														
?ENRLO	1127#	5362														
?EPACC	965#	2969	2975	3211	3217											
?EPPCH	1010#	2761	2777	2912	3354	3360										
?EPPCL	1001#	2734	2750	2806	2814	2819	3327	3333								
?EPPSW	974#	2792	2798	2839	2845	2894	2900	2939	2945	3249	3255	3284	3290			
?EPR0	992#	2924	2930	3268	3274	4655	4861	5060	5082							
?EPTIM	983#	2954	2960	3230	3236											
?FORM1	295#	1615	1634	1655	1688	1695	1722	1745	1780	1807	1819	1982	2000	2013	2178	2389
	2441	2445	2547	2587	2720	2735	2742	2762	2769	2793	2811	2818	2844	2862	2877	2899
	2911	2929	2944	2959	2974	3008	3069	3097	3212	3231	3250	3269	3289	3307	3328	3355
	3439	3458	3477	3496	3516	3531	3563	3584	3634	3638	3807	3814	3834	3841	3963	3987
	4002	4017	4043	4071	4263	4270	4290	4297	4322	4398	4439	4458	4550	4568	4593	4645
	4652	4765	4789	4804	4820	4842	4860	4876	4963	4987	5002	5018	5043	5061	5068	5096
	5168	5181	5197	5349	5361	5375	5387	5461	5504	5547	5581	5597	5616	5623	5693	5709
	5727	5808	5957	5971	6065	6084	6158	6229	6305							
?FORM2	319#	1638	1659	1692	1702	1986	2739	2749	2766	2776	2797	2815	2825	2866	3216	3235
	3254	3273	3311	3332	3359	3443	3462	3481	3500	3811	3821	3830	3848	3967	4267	4277
	4294	4304	4402	4649	4659	5065	5081	5465	5601	5620	5636	5713	6069			
?FORM3	339#	1755	2023	2452	2887	3541	3651	4053	4332	4449	4468	4560	4578	5520	5981	

All trademarks copyrighted © Intel Corporation 1978.

?R1	99#	4289	4305	4312	4312												
?RAM	103#	965	966	974	975	983	984	992	993	1001	1002	1010	1011	1019	1020	1028	
	1029	1037	1038	1046	1047	1055	1056	1064	1065	1073	1074	1082	1083	1091	1092	1100	
	1101	1109	1110	1118	1119	1127	1128	1136	1137	1145	1146	1154	1155	1163	1164	1172	
	1173	1181	1182	1190	1191	1199	1200	1208	1209	1217	1218	1226	1227	1235	1236	1244	
	1245	1253	1254	1262	1263	1271	1272	1280	1281	1289	1290	1298	1299				
?R0	101#	740	741	745	757	758	762	774	775	779	791	792	796	808	809	813	
	825	826	830														
?RB1	102#	849	850	854	858	870	871	875	879	891	892	896	900	912	913	917	
	921	933	934	938	942												
?RDLA	1244#	5688	5694	5708	5714	6064	6070	6079	6085								
?RECTV	1271#	3495	3501	3579	3585												
?REGC	1289#	4397	4403	4440	4450	4551	4561	4508	4594								
?ROTCN	870#																
?ROTPA	849#	5505	5512	5512	5521	5527	5527										
?RSAVE	144#	591	595	607	611	623	627	639	643	655	659	677	681	692	696	707	
	711	722	726	746	763	780	797	814	831	855	859	876	880	897	901	918	
	922	939	943	4223	4227												
?SEGMA	1300#																
?SIZE	255#	1383	1437	1861	1931	2141	2218	2284	2351	2502	2635	2662	3132	3378	3601	3669	
	3718	3753	3904	4090	4125	4164	4201	4343	4479	4603	4679	4708	4901	5122	5208	5248	
	5293	5397	5745	5824	5893	6017	6095	6168	6243	6318	6386	6459	6525	6596	6670	6743	
	6811	6869	6930	6991	7058	7118											
?SMAHI	1118#	2485	2485	2485	2491	2757	2765	2770	3457	3463	3802	3810	3815	4784	4790	4982	
	4988	5182	5370	5376													
?SMALO	1109#	2730	2730	2743	2861	2867	2870	2888	3306	3312	3476	3482	3829	3837	3842	4799	
	4805	4815	4821	4837	4843	4871	4877	4997	5003	5013	5019	5038	5044	5091	5097	5192	
	5198	5344	5350	6224	6230	6300	6306										
?START	1339#	1383	1383	1391	1405#	1437	1437	1445	1533#	1861	1861	1869	1882#	1931	1931	1939	
	1957#	2141	2141	2149	2162#	2218	2218	2226	2244#	2204	2204	2292	2310#	2351	2351	2359	
	2382#	2502	2502	2510	2533#	2635	2635	2643	2656#	2662	2662	2670	2699#	3132	3132	3140	
	3173#	3378	3378	3386	3414#	3601	3601	3609	3622#	3669	3669	3677	3695#	3718	3718	3726	
	3745#	3753	3753	3761	3796#	3904	3904	3912	3951#	4090	4090	4098	4116#	4125	4125	4133	
	4151#	4164	4164	4172	4190#	4201	4201	4209	4254#	4343	4343	4351	4385#	4479	4479	4487	
	4525#	4603	4603	4611	4635#	4679	4679	4687	4700#	4708	4708	4716	4754#	4901	4901	4909	
	4952#	5122	5122	5130	5150#	5208	5208	5216	5235#	5248	5248	5256	5279#	5293	5293	5301	
	5334#	5397	5397	5405	5449#	5745	5745	5753	5791#	5824	5824	5832	5865#	5893	5893	5901	
	5939#	6017	6017	6025	6053#	6095	6095	6103	6146#	6168	6168	6176	6220#	6243	6243	6251	
	6294#	6318	6318	6326	6371#	6386	6386	6394	6437#	6459	6459	6467	6512#	6525	6525	6533	
	6582#	6596	6596	6604	6653#	6670	6670	6678	6726#	6743	6743	6751	6773#	6811	6811	6819	
	6832#	6869	6869	6877	6890#	6930	6930	6938	6951#	6991	6991	6999	7048#	7058	7058	7066	
	7114#	7118	7118	7126													
?STRTH	1253#	1981	1987	1995	2001	2014	2024										
?TYPE	1172#	1577	1577	1577	1583	1746	1756	1769	1769	1769	1775	1820	2440	2446	2453	2542	
	2548	3003	3009	3064	3070	4760	4766	4958	4964	5163	5169						
?UNARY	459#	1744	2012	2076	3530	4042	4321	4430	4457	4549	4567	5503	5970				
?VERSN	1046#																
?XPCOD	825#																
?ZEKO	671#	1559	1575	1767	2483	3424	3856	5656									
AFETCH	4704	4759#															
ASAVE	1239#	5468	5730														
ASCERR	3704	3711	3715#														
B	1284#	4415	4421	4461	4529	4571											
BCODE	1167#	1563	1569	1598	1618	2392											
BITS0	4230#	4422															
BRKEND	2462	2500#															
BRKERR	3090	3080#															

All mnemonics copyrighted © Intel Corporation 1976.

Appendix C and D

APPENDIX C COMMAND SUMMARY

The following is a summary of the commands implemented by the HSE-49 emulator monitor. Within each command group, tokens in each column indicate options the user has when invoking those commands.

Tokens in square brackets indicate dedicated keys on the keyboard (some keys having shared functions); angle brackets enclose hex digit strings used to specify an address or data parameter. Parameters in parentheses are optional, with the effects explained above. The notation used is as follows:

<SMA> — Starting Memory Address for block command,
<EMA> — Ending Memory Address for block command,
<LOC> — LOcation for individual accesses,
<DATA> — DATA byte.

Asterisks (*) indicate the default condition for each command; thus that token is optional and serves to regularize the command syntax.

Program/data entry and verification commands:

```
[EXAM] [PROG MEM]* <LOC> [.] [NEXT]
        [DATA MEM]          [PREV]
        [REGISTER]          [.]
        [HWRE REG]
        [PROG BRK]
        [DATA BRK]
```

Program/data initialization commands:

```
[FILL] [PROG MEM]* <SMA> [.] <EMA> [.] <DATA> [.]
        [DATA MEM]
        [REGISTER]
        [HWRE REG]
        [PROG BRK]
        [DATA BRK]
```

Intellec® development system or TTY interface commands (for transferring HEX format files):

```
[UPLOAD] [PROG MEM]* <SMA> [.] <EMA> [.]
          [DATA MEM]
          [REGISTER]
          [HWRE REG]
          [PROG BRK]
          [DATA BRK]

[DNLOAD] [PROG MEM]* [.]
          [DATA MEM]
          [REGISTER]
          [HWRE REG]
          [PROG BRK]
          [DATA BRK]
```

Formatted data dump to TTY or CRT:

```
[LIST] [PROG MEM]* <SMA> [.] <EMA> [.]
        [DATA MEM]
        [REGISTER]
        [HWRE REG]
        [PROG BRK]
        [DATA BRK]
```

Program execution commands:

```
[GO]      [NO BREAK]* <SMA> [.]
          [W/ BREAK]          [.]
          [SING STP]
          [AUTO BRK]
          [AUTO STP]

[GO/RST]  [NO BREAK]* [.]
          [W/ BREAK]
          [SING STP]
          [AUTO BRK]
          [AUTO STP]
```

Breakpoint setting and clearing:

```
[SET BRK] [PROG MEM]* <LOC> ([.] <LOC> ... ) [.]
          [DATA MEM]

[CLR BRK] [PROG MEM]* <LOC> ([.] <LOC> ... ) [.]
          [DATA MEM]
```

APPENDIX D ERROR MESSAGES

The following error message codes are used by the monitor software to report an operator or hardware error. Errors may be cleared by pressing [CLR/PREV] or [END/]. The format used for reporting errors is "Error — .n" where "n" is a hex digit.

Operator Errors

1. Illegal command initiator.
2. Illegal command modifier or parameter digit.
3. Illegal terminator for Examine command.
4. Illegal attempt to clear Error mode.
- 5–9. Not used.

Hardware Errors

- A. ASCII error — non-hex digit encountered in data field of hex format record.
- B. Breakpoint error. Break logic activated though breakpoints not enabled.
- C. Hex format record checksum error. Note — the checksum will not be verified if the first character of the checksum field is a question mark ("?",) rather than a hexadecimal digit. This allows object files to be patched using the ISIS text editor without the necessity of manually recomputing the checksum value.
- D. Not used.
- E. Execution processor failed to respond to a command or parameter passed to it by the master processor. EP automatically reset. EP internal status may be lost. Program memory not affected.
- F. Not used.

Using the 8049 as an 80 Column Printer Controller

Contents

INTRODUCTION	3-188
PRINT MECHANISM DESCRIPTION	3-188
INTERFACE CIRCUITRY	3-189
SOFTWARE	3-191
HANDLING THE I/O BUFFER	3-192
TIMING	3-192
CONCLUSION	3-194
APPENDIX A. SCHEMATIC DIAGRAM	3-195
APPENDIX B. MONITOR LISTING	3-196

the printer is usually quite a bit different from a dot matrix printer, but they usually offer the print quality of character printers.

In recent years, the "computer boom" has caused the price of printers to tumble markedly. High volume desktop, competition, and the tremendous demand for reliable print mechanisms have all contributed to a decrease in price. Because of their simplicity, line printer mechanisms have decreased in price faster than other mechanisms. Therefore, when high quality print is not needed, a line printer is a very attractive choice.

This application note describes how to control an 80 column impact-line printer with an 8049. The complete software listing is included in the appendix. The 8049 is the high performance member of the MCS-48 microcontroller family. The Processor has all of the features of the 8048 plus twice the amount of program and data memory and an 11MHz clock speed. For details about the 8049, please refer to the MCS-48 user's manual.

II. PRINT MECHANISM DESCRIPTION

The model 830 printer is available from C. T. HOH ELECTRONICS (3301 BEEETHOVEN STREET, LOS ANGELES, CA 90068). This inexpensive and simple printer is ideal for applications requiring 80 columns of dot matrix alpha-numeric information.

The model 830 printer is comprised of three basic sub-assemblies: the chassis or frame, the paper feed mechanism, and the print head. The diagram in Figure 2.1 gives the physical dimensions of the basic print mechanism. The basic chassis for the printer is constructed out of four sheet metal stampings. These stampings are screwed together to form a sturdy base on which all other components of the printer are mounted.

The paper feed mechanism consists of a footed wheel, a solenoid, a tension spring, and a "catcher". When the solenoid is activated, the arm of the solenoid pulls against the spring and drags over the footed wheel. When the solenoid is released, its arm is pulled by the spring, but this time the arm grabs a tooth on the wheel and pulls the wheel forward which advances the paper. A "catcher", which is merely a piece of plastic held against the footed wheel, is added to assure that the paper is advanced only one "tooth" position each time the solenoid is activated.

The print head is comprised of seven solenoids which are mounted in a common housing. The solenoids are typically mounted in a circle, but their hammers are positioned linearly along the vertical axis. These seven vertically positioned hammers use the strikers that actually do the printing.

Over the last few years 80 column output devices have become somewhat of a de facto output standard for business and some data processing applications. It should be mentioned that by no means is the 80 column format a "new" standard. 80 column computer cards have been around for more than 20 years and perhaps the existence of these cards in the early days of computers is why the 80 column format is a standard today.

Many CPT terminals use the 80 by N format and to complement this a number of printers use this same format. One reason, aside from those historic in nature, for the 80 column standard is that 80 columns of 12 pitch text on standard typewritten 8.5 inch by 11 inch paper completely fill up an entire line and allow ample room for margins. So, the 80 column format is an aesthetically convenient format.

Printers are usually divided into either impact or non-impact and a character or line oriented device. Impact printers actually use some type of "striker" to place ink on the paper. More often than not the ink is contained on a ribbon which is placed between the striker and the paper. Non-impact printers use some means other than direct pressure to place the characters on the paper. This type of printer is very fast because there is very little mechanical motion associated with placing the characters on the paper. However, because the paper is required to be treated with a special substance, it is not as convenient as an impact printer.

Character printers are capable of printing one character at a time. (Any standard home typewriter is in effect a character printer.) Line printers must print an

USING THE 8049 AS AN 80 COLUMN PRINTER CONTROLLER

I. INTRODUCTION

This Application Note details using INTEL's 8049 microcomputer as a dot matrix printer controller. Previous INTEL Application notes, (e.g. AP-27 and AP-54) described using intelligent processors and peripherals to control single printer mechanisms. This Application note expands upon the theme established in these prior notes and extends the concept to include a complete bi-directional 80 column printer using a single line buffer. For convenience this application note is divided into six sections:

1. INTRODUCTION
2. PRINT MECHANISM DESCRIPTION
3. INTERFACE CIRCUITRY
4. SOFTWARE
5. CONCLUSION
6. APPENDIX

Over the last few years 80 column output devices have become somewhat of a defacto output standard for business and some data processing applications. It should be mentioned that by no means is the 80 column format a "new" standard. 80 column computer cards have been around for more than 20 years and perhaps the existence of these cards in the early days of computers is why the 80 column format is a standard today.

Many CRT terminals use the 80 by N format and to complement this a number of printers use this same format. One reason, aside from those historic in nature, for the 80 column standard is that 80 columns of 12 pitch text on standard typewritten 8.5 inch by 11 inch paper completely fills up an entire line and allow ample room for margins. So, the 80 column format is an aesthetically convenient format.

Printers are usually divided into either impact or non-impact and a character or line oriented device. Impact printers actually use some type of "striker" to place ink on the paper. More often than not the ink is contained on a ribbon which is placed between the striker and the paper. Non-impact printers use some means other than direct pressure to place the characters on the paper. This type of printer is very fast because there is very little mechanical motion associated with placing the characters on the paper. However, because the paper is required to be treated with a special substance, it is not as convenient as an impact printer.

Character printers are capable of printing one character at a time. (Any standard home typewriter is in effect a character printer.) Line printers must print an

entire line at a time. Line printers are usually quite a bit faster than character printers, but they usually don't offer the print quality of character printers.

In recent years, the "computer boom" has caused the price of printers to tumble markedly. High volume production, competition, and the tremendous demand for reliable print mechanisms have all contributed to the decrease in price. Because of their simplicity, line printer mechanisms have decreased in price faster than other mechanisms. Therefore, when high quality print is not needed, a line printer is a very attractive choice.

This application note describes how to control an 80 column impact-line printer with an 8049/8039. The complete software listing is included in the appendix. The 8049 is the high-performance member of the MCS-48TM microcontroller family. The Processor has all of the features of the 8048 plus twice the amount of program and data memory and an 11MHz clock speed. For details about the 8049, please refer to the MCS-48 user's manual.

II. PRINT MECHANISM DESCRIPTION

The model 820 printer is available from C. ITOH ELECTRONICS (5301 BEETHOVEN STREET, LOS ANGELES, CA 90066). This inexpensive and simple printer is ideal for applications requiring 80 columns of dot matrix alpha-numeric information.

The model 820 printer is comprised of three basic sub-assemblies; the chassis or frame, the paper feed mechanism, and the print head. The diagram in Figure 2.1 gives the physical dimensions of the basic print mechanism. The basic chassis for the printer is constructed out of four sheet metal stampings. These stampings are screwed together to form a sturdy base on which all other components of the printer are mounted.

The paper feed mechanism consists of a toothed wheel, a solenoid, a tension spring, and a "catcher." When the solenoid is activated, the arm of the solenoid pulls against the spring and drags over the toothed wheel. When the solenoid is released, its arm is pulled by the spring, but this time the arm grabs a tooth on the wheel and pulls the wheel forward which advances the paper. A "catcher," which is merely a piece of plastic held against the toothed wheel, is added to assure that the paper is advanced only one "tooth" position each time the solenoid is activated.

The print head is comprised of seven solenoids which are mounted in a common housing. The solenoids are physically mounted in a circle, but their hammers are positioned linearly along the vertical axis. These seven vertically positioned hammers are the strikers that actually do the printing.

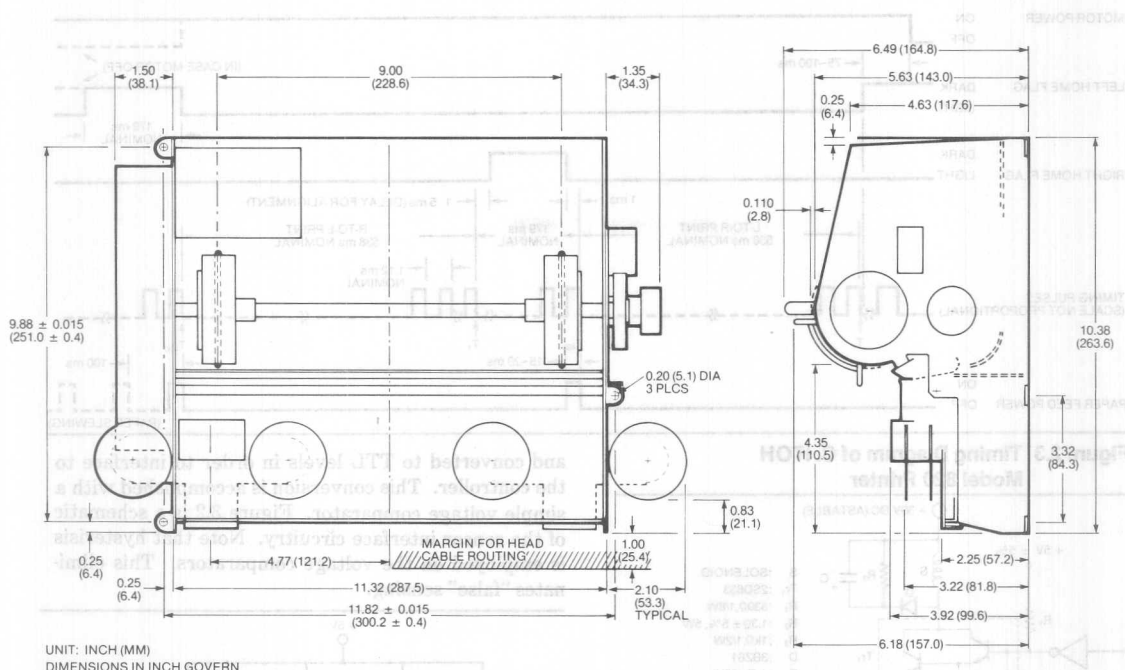


Figure 2.1 Physical Dimensions of C. ITOH Model 820 Printer

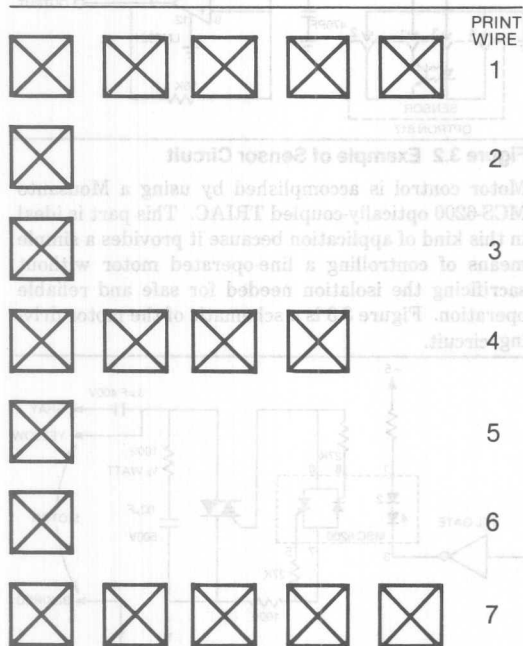


Figure 2.2 "Formation" of a Character by a Dot Matrix Printer

A motor, mounted toward the back of the print mechanism, drives a rubber toothed belt which turns a roller guide. A motor turns a guide that moves the print head from right to left and left to right. By properly timing the current flow through the solenoids while the print head is moving across the paper, characters can be formed. Figure 2.2 illustrates how the dot matrix printer "forms" its characters.

The timing pulses for the print head mechanism are generated by an opto-electronic sensor. This sensor, located on the left side plate of the printer, informs the print controller when to apply current to the print head mechanism. This "on-board timing wheel" assures that all characters will be properly spaced and that they will all be "in-line" in a vertical sense.

The print mechanism is also equipped with two additional sensors. These are the left home position sensor, located near the left front of the mechanism, and the right home position sensor, located near the right front of the print mechanism. These sensors simply tell the controller when the print head is in either the left or right home position. A complete timing chart for the printer is shown in Figure 2.3.

III. INTERFACE CIRCUITRY

The manual supplied with the printer recommends some specific interface circuitry. For the most part the circuitry used in this Application Note followed these suggestions. The circuitry needed to drive the print head solenoid is shown in Figure 3.1. This same

To interface 8049 to the outside world one 8212 latch was used. This latch was connected to the BUS PORT and is enabled by an INS or MOVX instruction coincident with BIT 4 of PORT 1 being in a logical zero state. In this configuration, the 8212 was used to hold the data until read by the 8049. The connection of the 8212 to the 8049 is shown in Figure 3.4 and the parallel port timing diagram is shown in Figure 3.5. The 8212 parallel port was connected to the LINE PRINTER OUTPUT of an INTELLEC MICROCOMPUTER DEVELOPMENT SYSTEM.

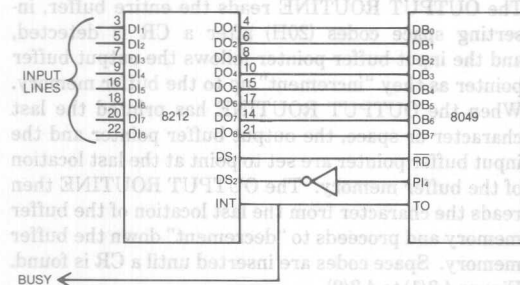


Figure 3.4 Connection of the 8212 Input Port to the 8049

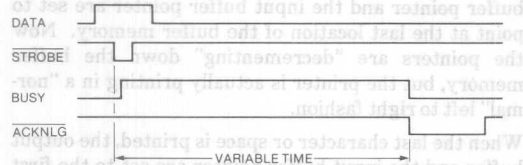


Figure 3.5 Parallel Port Timing

IV. SOFTWARE

As mentioned in Section 2, the bulk of the timing needed to control the printer is actually generated by the printer itself. Therefore, all the software must do is harness these timing signals and turn on and off the right solenoids at the right time.

To make things easy, the software needed to drive the printer is broken into four separate routines. These are:

1. INITIALIZATION ROUTINE
2. INPUT ROUTINE
3. OUTPUT ROUTINE
4. LOOKUP ROUTINE

The INITIALIZATION ROUTINE turns the motor on and checks the opto-electronic sensors. If a failure is found, the routine turns off the motor and loops on itself. This insures that the print mechanism is cycled properly before characters are accepted for printing.

This routine also initializes all of the variables used by the printer.

The INPUT ROUTINE reads the characters that are present in the 8212 input port and writes them into the 8049's buffer memory. The routine then checks the characters to see if a CARRIAGE RETURN (ASCII OCH) has been transmitted. If a CR is detected, the input routine automatically inserts a LINE FEED as the next character. When the input routine detects a LINE FEED, it stops reading characters and sets the direction bits and the print bit in the status register. This action evokes the OUTPUT ROUTINE. A detailed flowchart of the INPUT ROUTINE is shown in Figure 4.1.

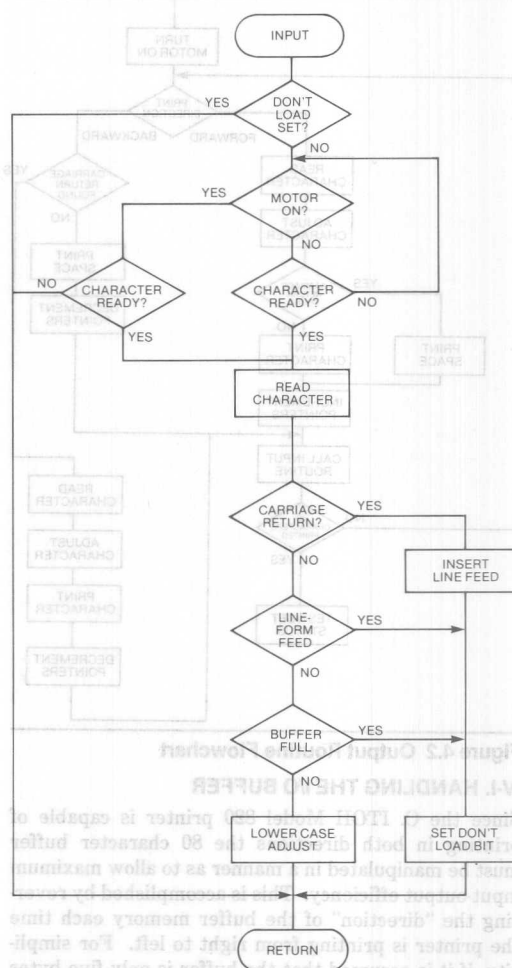


Figure 4.1 Input Routine Flowchart

The OUTPUT ROUTINE initializes both the input and output buffer pointers and then reads the characters from the 8049's buffer memory. After a character is read the OUTPUT ROUTINE calls the LOOKUP ROUTINE which reads the proper bit pattern to form that character. This bit pattern is then used to strobe the solenoids. After each character is printed, the OUTPUT ROUTINE calls the INPUT ROUTINE and another character is placed into the buffer memory. This type of operation guarantees that the input buffer cannot "overrun" the output buffer. A flowchart of the OUTPUT ROUTINE is shown in Figure 4.2.

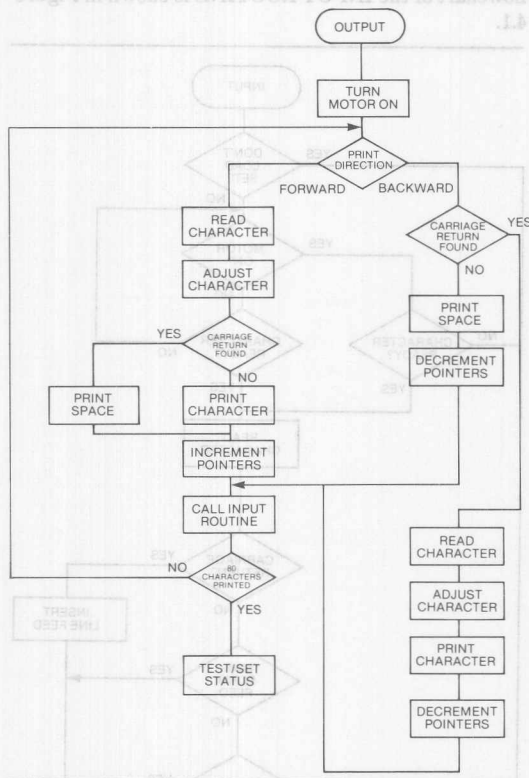


Figure 4.2 Output Routine Flowchart

IV-I. HANDLING THE I/O BUFFER

Since the C. ITOH Model 820 printer is capable of printing in both directions the 80 character buffer must be manipulated in a manner as to allow maximum input-output efficiency. This is accomplished by reversing the "direction" of the buffer memory each time the printer is printing from right to left. For simplicity, if it is assumed that the buffer is only five bytes long, Figure 4.3 can be used to help explain the buffer operation.

Initially the input buffer pointer is loaded with the address of the first location in the buffer memory. As characters are read, the input buffer pointer increments and fills the buffer memory as shown in Figure 4.3(b) through 4.3(f). When a CARRIAGE RETURN-LINE FEED (CRLF) is encountered the input buffer pointer and the output buffer pointer are reset back to the first location. The OUTPUT ROUTINE then reads the character from the first location in the buffer memory, increments the output buffer pointer and calls the INPUT ROUTINE, which reads another character from the parallel input port.

The OUTPUT ROUTINE reads the entire buffer, inserting space codes (20H) after a CR is detected, and the input buffer pointer follows the output buffer pointer as they "increment" up to the buffer memory. When the OUTPUT ROUTINE has printed the last character or space, the output buffer pointer and the input buffer pointer are set to point at the last location of the buffer memory. The OUTPUT ROUTINE then reads the character from the last location of the buffer memory and proceeds to "decrement" down the buffer memory. Space codes are inserted until a CR is found. Figure 4.3(1) to 4.3(0).

The input buffer pointer follows the output buffer pointer just as in the previous case. When the last, or in this case the first character is printed, the output buffer pointer and the input buffer pointer are set to point at the last location of the buffer memory. Now the pointers are "decrementing" down the buffer memory, but the printer is actually printing in a "normal" left to right fashion.

When the last character or space is printed, the output buffer and the input buffer pointer are set to the first location of the buffer memory and printing takes place in a reverse or right to left manner. After this line is printed, the print head and both buffer pointers are in the same position as they were initially. So, four lines must be printed before the buffer pointers and the print head complete a cycle. Each of these situations is handled separately by four different sub-routines: CASE0, CASE1, CASE2, and CASE3.

IV-II. TIMING

All critical timing for the printer controller came from two basic sources; the timing sensors on the printer and the internal eight-bit timer of the 8049.

The internal timer of the 8049 was used to control the length of time the solenoids were fired (600 microseconds) and was also used as a "one-shot" to align the printer. This alignment is needed to make the "backward" printing line up vertically with the normal or forward printing. The "one-shot" is used to measure the time from the last column of the last character position until the right sensor flag is covered.



Figure 4.3 I/O Buffer Handler

sensor flag is uncovered, the timer is then used to determine where to start printing in the reverse direction.

The timer and the print wheel on the printer are used to determine when to place a character. The strobe from the print wheel informs the 8049 when to fire the solenoids and the timer allows the proper spacing between the characters.

V. CONCLUSION

Although the full speed of the 8049 was not used in this application, the high speed of the 8049 makes it possible to "fine-tune" any critical timing parameters. Additionally, the extra available CPU time could be used to add an interrupt driven keyboard and display, such as the ones discussed in AP-40, to the printer. This would allow the printer to function as a complete "terminal".

Very little attempt was made to optimize the software, but still the entire program fits easily in 1.25K of memory; 750 bytes for printer control and 500 bytes for character lookup. Adding lower case to the printer would require an additional 500 bytes of lookup table. The remaining 250 bytes should be used to add "user" features such as tabs, double width printing, etc.

The high speed of the 8049 combined with its hardware and software architecture make it an ideal choice for controlling an 80 column, bi-directional line printer. The I/O structure of the 8049 minimizes the amount of external hardware needed to control the printer and the large amount of on-board program and data memory allow quite a sophisticated control program to be implemented.

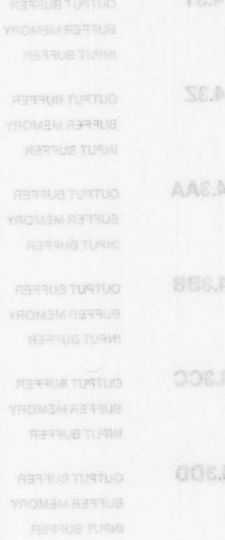
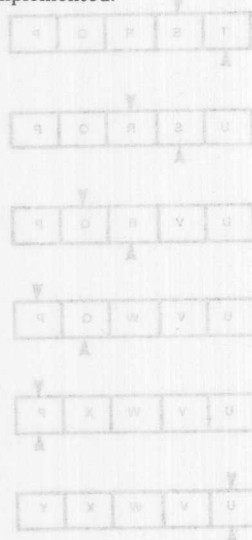
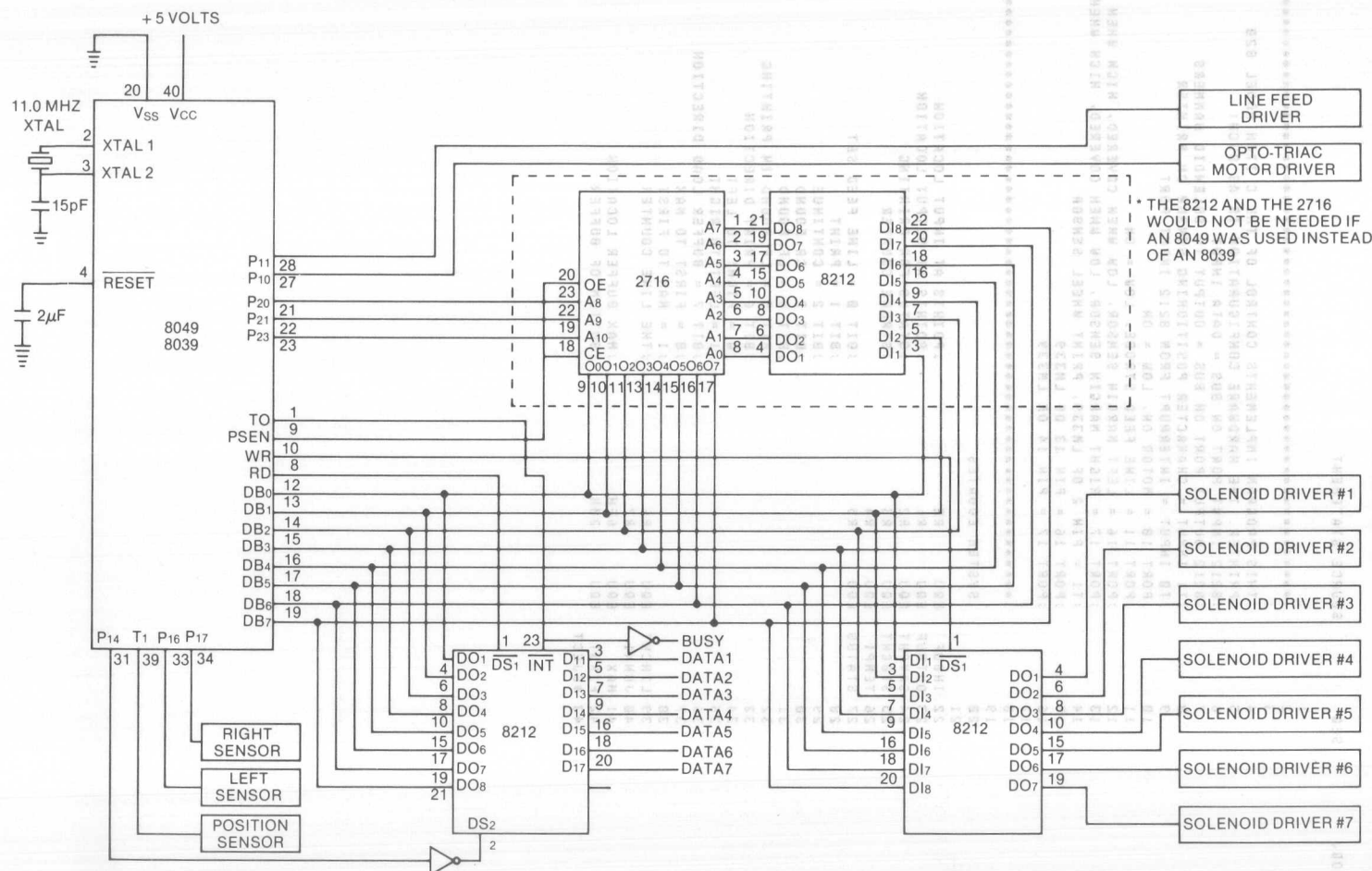


Figure 4.3 NO Buffer Handler

3-195



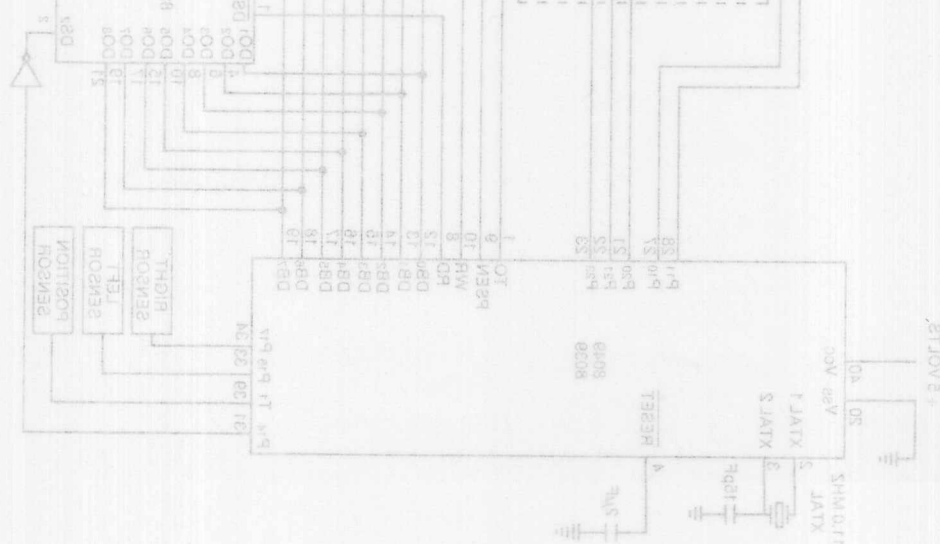
APPENDIX B. MONITOR LISTING

```

LOC 0B0  SEQ  SOURCE STATEMENT
1  *****
2  THIS PROGRAM IMPLEMENTS CONTROL OF THE C. ITOH MODEL 820
3  PRINTER. THE HARDWARE CONFIGURATION IS AS SUCH:
4  8212 INPUT PORT ON BUS = DATA INPUT
5  8212 OUTPUT PORT ON BUS = OUTPUT TO SOLENOID HAMMERS
6  T1 INPUT = CHARACTER POSITIONING SENSOR ON PRINTER
7  T0 INPUT = INTERRUPT FROM 8212 INPUT PORT
8  PORT 10 = MOTOR ON, LOW = ON
9  PORT 11 = LINE FEED STROBE, LOW = ON
10 PORT 16 = LEFT MARGIN SENSOR, LOW WHEN COVERED, HIGH WHEN OPEN
11 PORT 17 = RIGHT MARGIN SENSOR, LOW WHEN COVERED, HIGH WHEN OPEN
12 T1 = PIN 2 OF LM339, PRINT WHEEL SENSOR
13 PORT 16 = PIN 13 OF LM339
14 PORT 17 = PIN 14 OF LM339
15 *****
16
17 SYSTEM EQUATES
18
19 22 INBUF EQU R0 ;POINTS AT INPUT LOCATION
20 23 OUTBUF EQU R1 ;POINTS AT OUTPUT LOCATION
21 24 SAVPNT EQU R2 ;STATUS FOR PRINTING
22 25 STBCNT EQU R3 ;STROBE COUNTER
23 26 TEMP1 EQU R4
24 27 STATUS EQU R5
25
26 ;BIT 0 = LINE FEED SET
27 ;BIT 1 = PRINT
28 ;BIT 2 = CONTINUE
29 ;BIT 3 = CR FOUND
30 ;BIT 4 = LF FOUND
31 ;BIT 5 = LF FOUND IN PRINTING
32 ;BIT 6 = PRINT DIRECTION
33 ;B = RIGHT TO LEFT
34 ;L = LEFT TO RIGHT
35 ;BIT 7 = BUFFER LOAD DIRECTION
36 ;B = FIRST TO MAX
37 ;L = MAX TO FIRST
38 ;THE LINE COUNTER
39
40 LINCHT EQU R6
41 JUNK1 EQU R7
42 MAX EQU 6FH ;MAX BUFFER LOCATION
43 FIRST EQU 20H ;BOTTOM OF BUFFER
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

APPENDIX B SCHEMATIC DIAGRAM



LOC	OBJ	SEQ	SOURCE STATEMENT	THINSTATE 338802	032	LOC	OBJ
		44	RTN: END INT TSC	RTN: END INT TSC	032	17	1000
0000		45	ORG 0000 0000	ORG 0000 0000	032	18	1000
		46	RTN: END INT TSC	RTN: END INT TSC	032	19	1000
		47	JUMP OVER THE INTERRUPT LOCATIONS	JUMP OVER THE INTERRUPT LOCATIONS	032	20	1000
		48	RTN: END INT TSC	RTN: END INT TSC	032	21	1000
0000 15		49	DIS: NOT I TSC	DIS: NOT I TSC	032	22	1000
0001 0400		50	JMP 0000 0000	JMP 0000 0000	032	23	1000
		51	RTN: END INT TSC	RTN: END INT TSC	032	24	1000
000A		52	ORG 0000 0000	ORG 0000 0000	032	25	1000
		53	RTN: END INT TSC	RTN: END INT TSC	032	26	1000
		54	START THE PROGRAM	START THE PROGRAM	032	27	1000
		55	RTN: END INT TSC	RTN: END INT TSC	032	28	1000
		56	LOOP UNTIL THE BUFFER FILLS UP	LOOP UNTIL THE BUFFER FILLS UP	032	29	1000
		57	RTN: END INT TSC	RTN: END INT TSC	032	30	1000
000A FD		58	PRNT: MOV 0000 A: STATUS	PRNT: MOV 0000 A: STATUS	032	31	1000
000B 3211		59	JB 0000 0000	JB 0000 0000	032	32	1000
000D 3400		60	CALL T: LDBUF	CALL T: LDBUF	032	33	1000
000F 040A		61	JMP PRNT: LOOP	JMP PRNT: LOOP	032	34	1000
		62			032	35	1000
		63	THIS ROUTINE PRINTS A LINE	THIS ROUTINE PRINTS A LINE	032	36	1000
		64	IT FIRST SAVES THE STATUS	IT FIRST SAVES THE STATUS	032	37	1000
		65	AND THEN DETERMINES WHICH DIRECTION TO PRINT	AND THEN DETERMINES WHICH DIRECTION TO PRINT	032	38	1000
		66	AND HOW TO MANIPULATE THE BUFFER	AND HOW TO MANIPULATE THE BUFFER	032	39	1000
		67	RTN: END INT TSC	RTN: END INT TSC	032	40	1000
0011 0409		68	LPRT: JMP 0000 0000	LPRT: JMP 0000 0000	032	41	1000
0013 F224		69	LPRT: JB 0000 0000	LPRT: JB 0000 0000	032	42	1000
0015 0417		70	JMP 0000 0000	JMP 0000 0000	032	43	1000
		71			032	44	1000
		72	CASE 01: LOADING THE BUFFER FROM FIRST TO MAX	CASE 01: LOADING THE BUFFER FROM FIRST TO MAX	032	45	1000
		73			032	46	1000
0017 0920		74	CASE 01: MOV 0000 OUTBUF, #FIRST	CASE 01: MOV 0000 OUTBUF, #FIRST	032	47	1000
0019 0820		75	MOV INBUF, #FIRST	MOV INBUF, #FIRST	032	48	1000
001B FA		76	MOV 0000 A: SAVPNT	MOV 0000 A: SAVPNT	032	49	1000
001C 940C		77	CALL MOTON	CALL MOTON	032	50	1000
001E D252		78	JB 0000 0000	JB 0000 0000	032	51	1000
0020 94B3		79	CALL PRINTBK	CALL PRINTBK	032	52	1000
0022 0431		80	JMP 0000 0000	JMP 0000 0000	032	53	1000
		81	RTN: END INT TSC	RTN: END INT TSC	032	54	1000
		82	CASE 23: LOADING BUFFER FROM MAX TO FIRST	CASE 23: LOADING BUFFER FROM MAX TO FIRST	032	55	1000
		83			032	56	1000
0024 096F		84	CASE 23: MOV OUTBUF, #MAX	CASE 23: MOV OUTBUF, #MAX	032	57	1000
0026 086F		85	MOV INBUF, #MAX	MOV INBUF, #MAX	032	58	1000
0028 FA		86	MOV A: SAVPNT	MOV A: SAVPNT	032	59	1000
0029 940C		87	CALL MOTON	CALL MOTON	032	60	1000
002B D2C2		88	JB 0000 0000	JB 0000 0000	032	61	1000
002D 94B3		89	CALL PRINTBK	CALL PRINTBK	032	62	1000
002F 043D		90	JMP 0000 0000	JMP 0000 0000	032	63	1000
		91			032	64	1000
		92	*EJECT	*EJECT	032	65	1000

0032 3471	94	CALL	FXPRNT	;ADJUST FOR PRINTING	0000
0034 B120	95	MOV	@OUTBUF,#20H	;PUT A SPACE IN BUFFER RAM	
0036 F242	96	JB7	FDC1	;FOUND A CR	
0038 945E	97	CALL	INCTST	;UPDATE OUTBUF	
003A C6AE	98	JZ	WATCHD	;WAIT FOR END	01 0000
003C BF20	99	MOV	JUNK1,#20H	;GET A SPACE TO PRINT	0000 1000
003E 9463	100	CALL	GTPRNT	;GO PRINT A SPACE	
0040 B431	101	JMP	CASE0	;LOOP	0000
0042 BF20	102 FDC1:	MOV	JUNK1,#20H	;GO PRINT THE LAST SPACE	
0044 9463	103 FDC1:	CALL	GTPRNT	;GO PRINT A CHARACTER	
0046 945E	104	CALL	INCTST	;CHECK OUT BUFFER	
0048 C6AE	105	JZ	WATCHD	;WAIT FOR THE END	
004A F1	106	MOV	A,@OUTBUF	;GET THE CHARACTER	
004B B120	107	MOV	@OUTBUF,#20H	;PUT A SPACE THERE	07 0000
004D 3491	108	CALL	FXPRNT	;FIX THE CHARACTER UP	1100 0000
004F AF	109	MOV	JUNK1,A	;SAVE IT	0000 0000
0050 B444	110	JMP	FDC1	;LOOP	0000 1000
	111				
	112				
	113			;CASE 1, PRINTING LEFT TO RIGHT, LOADING BUFFER FROM	
	114			;FIRST TO MAX	
	115				
0052 F1	116 CASE1:	MOV	A,@OUTBUF	;GET THE CHARACTER	
0053 3491	117	CALL	FXPRNT	;ADJUST FOR PRINTING	0000 1100
0055 AF	118	MOV	JUNK1,A	;SAVE ACC	0000 1100
0056 B120	119	MOV	@OUTBUF,#20H	;PUT A SPACE IN THE BUFFER	1100 0100
0058 F262	120	JB7	CRFOND	;FOUND A CR?	
005A 9463	121	CALL	GTPRNT	;GO PRINT THE CHARACTER	
005C 945E	122	CALL	INCTST	;CHECK THE BUFFER	
005E C675	123	JZ	WATCH	;IS THE LAST CHARACTER BEING PRINTED?	0000 1100
0060 B452	124	JMP	CASE1	;LOOP	0000 0100
0062 B120	125 CRFOND:	MOV	@OUTBUF,#20H	;PUT A SPACE IN THE BUFFER MEMORY	0000 0100
0064 BF20	126	MOV	JUNK1,#20H	;PUT A SPACE IN TEMP LOCATION	0000 0100
0066 9463	127	CALL	GTPRNT	;GO PRINT THE SPACE	0000 0100
0068 945E	128	CALL	INCTST	;CHECK THE BUFFER	0000 0100
006A C675	129	JZ	WATCH	;LAST CHARACTER PRINTED?	0000 0100
006C F1	130	MOV	A,@OUTBUF	;GET THE NEXT CHARACTER	
006D 3491	131	CALL	FXPRNT	;ADJUST IT	
006F B462	132	JMP	CRFOND	;LOOP	
	133 #EJECT				

LOC	OBJ	SEQ	SOURCE STATEMENT	1X3M3T4T2 338002	Q32	422	361
		184	;				
		185	;THIS ROUTINE WAITS FOR THE SENSOR FLAGS TO BE COVERED				
		186	;WHEN PRINTING RIGHT TO LEFT				
		187	;				
BB8E	348B	188	WATCHD: CALL LDBUF ;GO READ THE LAST CHARACTER				
BB8B	B9	189	IN A,P1 ;GET SENSOR INFORMATION				
BB81	D2AE	190	JB6 WATCHD ;LOOP IF SENSOR IS NOT COVERED				
BB83	FD	191	MOV A,STATUS ;GET THE STATUS				
BB84	528A	192	JB2 DVB ;SEE IF CONTINUE IS SET				
BB86	94DF	193	CALLA MOTOF ;TURN THE MOTOR OFF				
BB88	53FD	194	ANL A,#BFDH ;RESET BIT 1				
BB8A	53FB	195	ANL A,#BFBH ;RESET BIT 3				
BB8C	AD	196	MOV STATUS,A ;RESTORE STATUS				
BB8D	FA	197	MOV A,SAYPNT ;GET THE SAVED STATUS				
BB8E	B271	198	JB5 DOLF ;DO A LINE FEED				
BB8B	B48A	199	JMP PRNT ;EXIT				
		200	;				
		201	;CASE 3, PRINTING LEFT TO RIGHT, LOADING BUFFER FROM				
		202	;MAX TO FIRST				
		203	;				
BB82	F1	204	CASE3: MOV A,OUTBUF ;GET A CHARACTER				
BB83	3491	205	CALL FXPRNT ;FIX FOR PRINTING				
BB85	AF	206	MOV JUNK1,A ;SAVE CHARACTER				
BB86	B12B	207	MOV A,#20H ;PUT A SPACE IN THE BUFFER				
BB88	F2D2	208	JB7 CRFND ;LEAVE IF A CR IS FOUND				
BB8A	9463	209	CALL GTPRNT ;GO PRINT THE CHARACTER				
BB8C	9472	210	CALL DECTST ;CHECK THE BUFFER				
BB8E	C675	211	JZ WATCH ;LEAVE IF DONE				
BB8B	B4C2	212	JMP CASE3 ;LOOP				
BB82	B12B	213	CRFND: MOV A,OUTBUF,#20H ;PUT A SPACE IN THE BUFFER RAM				
BB84	BF2B	214	MOV JUNK1,#20H ;GET A SPACE				
BB86	9463	215	CALL GTPRNT ;PRINT A SPACE				
BB88	9472	216	CALL DECTST ;CHECK THE BUFFER				
BB8A	C675	217	JZ WATCH ;LEAVE IF DONE				
BB8C	F1	218	MOV A,OUTBUF ;GET NEXT CHARACTER				
BB8D	3491	219	CALL FXPRNT ;ADJUST IT				
BB8F	B4D2	220	JMP CRFND ;LOOP				
		221	EJECT				

LOC	OBJ	SEQ	SOURCE STATEMENT	TRANSLATED	ADDR	DISP
B100		222	ORG 100H			
B100 B9		224	LDBUF: IN A,P1	:READ PORT 1		
B101 B21C		225	JB5 LMODE	:BIT 5 = H = LINE MODE		
B103 12B7		226	JBB ARND	:JUMP AROUND IF MOTOR IS ON		
B105 89B1		227	ORL P1,#01H	:TURN THE MOTOR OFF		
B107 92B8		228	ARND: JB4 NOFF	:NO FORM FEED		
B109 FE		229	MOVH A,LINCN	:GET THE LINE COUNTER		
B10A 43B8		230	ORL A,#0BH	:SET MSB		
B10C AE		231	MOV A,LINCN	:RESTORE THE LINE COUNTER		
B10D 23FF		232	MOV A,#0FFH	:SET ACC		
B10F 721A		233	NOFF: JB3 NOLF	:JUMP IF NO LINE FEED		
B111 9478		234	CALL LINEFD	:GO DO A LF OR FF		
B113 B9		235	BUTLOP: INI BT,P1	:READ THE PORT		
B114 721A		236	JB3 NOLF	:WAIT FOR SWITCH TO BE RELEASED		
B116 921A		237	JB4 NOLF	:WAIT FOR SWITCH TO BE RELEASED		
B118 2413		238	JMP BUTLOP	:LOOP		
B11A 24B8		239	NOLF: JMP LDBUF	:LOOP		
		240				
		241	BT: BT	:FIRST SEE IF A CHARACTER IS PRESENT IN THE BUFFER		
		242				
B11C 261F		243	LMODE: JNB CHAR	:IF CHARACTER PRESENT, READ IT		
B11E B3		244	RET	:IF NOT, EXIT ROUTINE		
		245				
		246	BT: BT	:IF THERE IS A CHARACTER, READ IT		
		247				
B11F FD		248	CHAR: MOVH A,STATUS	:GET THE STATUS		
B120 5249		249	JB2 ARNDJP	:IF CONTINUE IS SET, DON'T LOAD		
B122 9249		250	JB4 ARNDJP	:IF LF IS SET, DON'T LOAD		
B124 724A		251	MUL: JB3 LFCRCK	:WAS CR SET, SEE IF NEXT CHAR IS LF		
B126 94D6		252	CALL GTCAR	:GO READ A CHARACTER		
B128 3461		253	GOOD: CALL FXCHAR	:MAKE SURE IT IS OK		
B12A AB		254	MOVH A,INBUF	:SAVE CHARACTER IN BUFFER MEMORY		
B12B FD		255	MOVH A,STATUS	:GET THE STATUS		
B12C F239		256	JB7 SUB1	:IF BIT 7 IS SET DECREMENT BUFFER		
B12E 18		257	INC INBUF	:UPDATE INBUF		
B12F 237B		258	MOV A,#MAX+1	:GET TOP		
B131 D8		259	XRL A,INBUF	:ARE WE AT THE TOP?		
B132 9649		260	JNZ ARNDJP	:IF NOT GET THE STATUS		
B134 F8		261	MOV A,INBUF	:GET INBUF		
B135 B7		262	DEC A	:CHANGE BY ONE		
B136 AB		263	MOV INBUF,A	:PUT IT BACK		
B137 2449		264	JMP ARNDJP	:GET THE STATUS		
B139 F8		265	SUB1: MOV A,INBUF	:GET INBUF		
B13A B7		266	DEC A	:CHANGE BY ONE		
B13B AB		267	MOV INBUF,A	:PUT INBUF BACK		
B13C 231F		268	MOV A,#FIRST-1	:GET THE BOTTOM OF THE BUFFER		
B13E D8		269	XRL A,INBUF	:TEST THE BUFFER		
B13F 9649		270	JNZ ARNDJP	:IF NOT ZERO READ THE STATUS		
B141 18		271	INC INBUF	:MOVE INBUF BACK		
B142 2449		272	JMP ARNDJP	:GO GET STATUS		
B144 FD		273	GETSTA: MOV A,STATUS	:GET THE STATUS		
B145 1249		274	JB8 ARNDJP	:IF BIT 8 SET, BYPASS		
B147 925B		275	JB4 STBIT1	:IF LF IS FOUND, SET THE STATUS		
B149 B3		276	ARNDJP: RET	:EXIT		
		277				
		278	BT: BT	:THIS ROUTINE "FORCES" A LF AFTER A CR		
		279				
B14A 94D6		280	LFCRCK: CALL GTCAR	:READ A CHARACTER		
B14C 23BA		281	MOV A,#BAH	:GET A LINE FEED		
B14E 2428		282	JMP GOOD	:JUMP BACK		
		283				
		284	BT: BT	:THIS ROUTINE SETS THE STATUS BITS		
		285				
B150 FD		286	STBIT1: MOV A,STATUS	:LOAD THE STATUS		
B151 3259		287	JB12 STPRNT	:IF STILL PRINTING, LEAVE		
B153 43B2		288	ORL A,#02H	:SET PRINT BIT		
B155 B34B		289	ADD A,#0BH	:UPDATE POSITION COUNTER		
B157 AD		290	MOVH A,STATUS	:PUT STATUS BACK		
B158 B3		291	RET	:EXIT ROUTINE		
B159 526B		292	STPRNT: JB2 BYEBYE	:CHECK CONTINUE BIT		
B15B 43BA		293	ORL A,#04H	:SET CONTINUE BIT		
B15D B34B		294	ADD A,#0BH	:UPDATE PRINT DIRECTION		
B15F AD		295	MOVH A,STATUS	:PUT THE STATUS BACK		
B160 B3		296	BYEBYE: RET	:EXIT		
		297				


```

000000      388          JUMPER CASE
B161 97       389 FRCHAR: CLR C ;CLEAR THE CARRY
B162 537F     390 ANL A,#7FH ;STRIP MSB
B164 AF       391 MOV JUNK1,A ;SAVE ACC
B165 B3AB     392 ADD A,#BABH ;SEE IF NUMBER IS 6BH
B167 E67B     393 JNC FINE ;IF CARRY ISN'T SET, JUMP
B169 FF       394 MOV A,JUNK1 ;GET ACC BACK
B16A 37       395 CPL A ;SUBTRACT 2BH FROM THE ACC
B16B B32B     396 ADD A,#2BH
B16D 37       397 CPL A
B16E 2474     398 JMP FIXDUN ;JUMP TO TEST CR LF
B17B 37       399 FINE: CPL A ;NOW SUBTRACT ABH FROM ACC
B171 B3AB     400 ADD A,#BABH
B173 37       401 CPL A
B174 AF       402 FIXDUN: MOV JUNK1,A ;SAVE A
B175 D3BD     403 XRL A,#BDH ;IS CHARACTER A CR
B177 967F     404 JNZ LFTST ;IF IT IS NOT TEST LF
B179 FD       405 MOV A,STATUS ;GET THE STATUS
B17A 43BB     406 ORL A,#BSH ;SET BIT 3
B17C AD       407 MOV STATUS,A ;RESTORE THE STATUS
B17D 24BF     408 JMP FIXFIN ;LEAVE
B17F FF       409 LFTST: MOV A,JUNK1 ;GET CHARACTER BACK
B180 D3BA     410 XRL A,#BAH ;IS IT A LF
B182 C6B9     411 JZ FIXUP ;IF ITS NOT, WE ARE DONE
B184 FF       412 MOV A,JUNK1 ;GET THE CHARACTER BACK
B185 D3BC     413 XRL A,#BCH ;IS IT A FORM FEED
B187 96BF     414 JNZ FIXFIN ;IF NOT FORM FEED, JUMP
B189 FD       415 MOV A,STATUS ;GET THE STATUS
B18A 431B     416 ORL A,#1BH ;SET BIT 4
B18C AD       417 MOV STATUS,A ;RETURN THE STATUS
B18D 345B     418 CALL STBIT1 ;SET THE STATUS
B18F FF       419 FIXFIN: MOV A,JUNK1 ;GET THE CHARACTER
LOC OBJ     420 SEQ SOURCE STATEMENT

B19B 83       421 RET ;EXIT FIXCHAR
;
;THIS ROUTINE RECOGNIZES A LF, FF, AND CR
;DURING THE PRINT OPERATION
;IT ALSO FORCES A SPACE IF A CHARACTER FOUND
;IN THE BUFFER IS NOT IN THE LOOKUP TABLE
B191 AF       422 FXPRNT: MOV JUNK1,A ;SAVE ACC
B192 D3BC     423 XRL A,#BCH ;FORM FEED
B194 C6B2     424 JZ FFFIX ;GO SET FORM FEED
B196 FF       425 MOV A,JUNK1 ;RESTORE CHARACTER
B197 D3BD     426 XRL A,#BDH ;SEE IF IT IS A CR
B199 C6AB     427 JZ CRFIX ;LEAVE IF IT IS
B19B FF       428 MOV A,JUNK1 ;GET ACC BACK
B19C D3BA     429 XRL A,#BAH ;SEE IF IT IS A LF
B19E C6AB     430 JZ LFFIX ;LEAVE IF IT IS
B1A0 FF       431 MOV A,JUNK1 ;GET CHARACTER BACK
B1A1 53EB     432 ANL A,#BEBH ;SEE IF IT IS A CHARACTER
B1A3 96BD     433 JNZ ISCHAR ;IF IT IS JUMP
B1A5 232B     434 MOV A,#2BH ;PUT A SPACE IN ACC
B1A7 83       435 RET ;EXIT
B1A8 43BB     436 CRFIX: ORL A,#BSH ;SET BIT 7
B1AA 83       437 RET ;EXIT
B1AB FD       438 LFFIX: MOV A,STATUS ;GET THE STATUS
B1AC 432B     439 ORL A,#2BH ;SET LF BIT IN STATUS
B1AE AD       440 MOV STATUS,A ;PUT THE STATUS BACK
B1AF 232B     441 MOV A,#2BH ;GET A SPACE
B1B1 83       442 RET ;EXIT
B1B2 FD       443 FFFIX: MOV A,STATUS ;GET THE STATUS
B1B3 432B     444 ORL A,#2BH ;SET LINE FEED BIT
B1B5 AD       445 MOV STATUS,A ;PUT THE STATUS BACK
B1B6 FE       446 MOV A,LINCNT ;GET THE LINE COUNT
B1B7 43BB     447 ORL A,#BSH ;SET BIT 7
B1B9 AE       448 MOV LINCNT,A ;PUT LINE COUNT BACK
B1BA 232B     449 MOV A,#2BH ;GET A SPACE
B1BC 83       450 RET ;EXIT
B1BD FF       451 ISCHAR: MOV A,JUNK1 ;GET CHARACTER BACK
B1BE 533F     452 ANL A,#3FH ;STRIP THE TWO MSB
B1CB 83       453 RET ;EXIT

```

LOC	OBJ	SEQ	SOURCE STATEMENT	THIRSTATE 23802	932	100	303
		371	;				
		372	;				
		373	;				
B1C1 AC		374	PRNTIT: MOV TEMP1, A	;	SAVE CHARACTER		
B1C2 E7		375	RL A	;	MULTIPLY BY TWO		
B1C3 E7		376	RL A	;	MULTIPLY BY FOUR		
B1C4 6C		377	ADD A, TEMP1	;	ADD ONCE TO MULTIPLY BY 5		
		378	;				
		379	;				
		380	;				
B1C5 2C		381	XCH A, TEMP1	;	PUT CHARACTER IN A, TARGET IN TEMP1		
B1C6 B2CA		382	JBS SHORT	;	JUMP TO HIGH ADDRESS IF BIT 5 SET		
B1C8 44AB		383	JMP PAGE1	;	GO TO FIRST PART OF LOOKUP TABLE		
B1CA 64AB		384	JMP PAGE2	;	GO TO SECOND PAGE OF LOOKUP TABLE		
		385	;				
		386	;				
		387	;				
		388	;				
B1CC AF		389	FIRE: MOV JUNK1, A	;	SAVE THE ACC		
B1CD FD		390	MOV A, STATUS	;	GET THE STATUS		
B1CE D2D4		391	JBG NT1	;	SEE IF FORWARD OR BACKWARDS		
B1D0 56D0		392	FIREX: JTI FIREX	;	WAIT FOR T1		
B1D2 24D6		393	JMP FIREY	;	LEAVE		
B1D4 46D4		394	NT1: JNT1 NT1	;	LOOP		
B1D6 FF		395	FIREY: MOV A, JUNK1	;	GET ACC BACK		
B1D7 9B		396	MOVX PRB, A	;	TRIGGER THE SOLENOID		
		397	;				
		398	;				
		399	;				
B1D8 23F3		400	MOV A, #BF3H	;	LOAD DELAY NUMBER		
B1DA 62		401	MOV T, A	;	PUT IT IN TIMER		
B1DB 55		402	STRT T	;	START THE TIMER		
B1DC 16EB		403	TSJTF: JTF KTDUN	;	LOOP ON TIMER FLAG		
B1DE 24DC		404	JMP TSJTF	;			
B1EB 27		405	KTDUN: CLR A	;	ZERO ACC		
B1E1 9B		406	MOVX PRB, A	;	TURN OFF SOLENOIDS		
B1E2 65		407	STOP TCNT	;	STOP THE TIMER		
B1E3 83		408	RET	;	EXIT FIRE ROUTINE		
		409	\$EJECT				

LOC	OBJ	SEQ	SOURCE STATEMENT
		410	;
		411	;
		412	;
		413	THIS IS THE LOOKUP TABLE. THE MSB IS NOT USED. THE MSB - 1
		414	IS THE DOT THAT IS THE TOP OF ANY GIVEN CHARACTER AND THE
		415	LSB IS THE DOT THAT IS THE BOTTOM OF ANY GIVEN CHARACTER
		416	;
		417	;
		418	;
B200		419	ORG 200H
		420	;
B200 3E		421	TABLE1: DB 3EH
B201 41		422	DB 41H
B202 5D		423	DB 5DH
B203 59		424	DB 59H
B204 4E		425	DB 4EH
		426	;
B205 7C		427	DB 7CH
B206 12		428	DB 12H
B207 11		429	DB 11H
B208 12		430	DB 12H
B209 7C		431	DB 7CH
		432	;
B20A 7F		433	DB 7FH
B20B 49		434	DB 49H
B20C 49		435	DB 49H
B20D 49		436	DB 49H
B20E 36		437	DB 36H
		438	;
B20F 3E		439	DB 3EH
B210 41		440	DB 41H
B211 41		441	DB 41H
B212 41		442	DB 41H
B213 22		443	DB 22H
		444	;
B214 7F		445	DB 7FH
B215 41		446	DB 41H
B216 41		447	DB 41H
B217 41		448	DB 41H
B218 3E		449	DB 3EH
		450	;
B219 7F		451	DB 7FH
B21A 49		452	DB 49H
B21B 49		453	DB 49H
B21C 49		454	DB 49H
B21D 41		455	DB 41H
		456	\$EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT	TABSTAT2 334002	Q32	LOG 343
		457				
B21E	7F	458	DB **** 7FH	: ***** 00	010	00 0000
B21F	09	459	DB * 09H	: ***** 00	010	10 0000
B220	09	460	DB * 09H	: ***** 00	010	10 0000
B221	09	461	DB * 09H	: ***** 00	010	10 0000
B222	01	462	DB **** 01H	: ***** 00	010	30 0000
		463				
B223	3E	464	DB **** 3EH	: ***** 00	010	00 0000
B224	41	465	DB * 41H	: ***** 00	010	00 0000
B225	41	466	DB * 41H	: ***** 00	010	00 0000
B226	51	467	DB * 51H	: ***** 00	010	00 0000
B227	71	468	DB * 71H	: ***** 00	010	00 0000
		469				
B228	7F	470	DB **** 7FH	: ***** 00	010	00 0000
B229	08	471	DB * 08H	: ***** 00	010	00 0000
B22A	08	472	DB * 08H	: ***** 00	010	00 0000
B22B	08	473	DB * 08H	: ***** 00	010	00 0000
B22C	7F	474	DB **** 7FH	: ***** 00	010	00 0000
		475				
B22D	08	476	DB **** 08H	: ***** 00	010	00 0000
B22E	41	477	DB * 41H	: ***** 00	010	00 0000
B22F	7F	478	DB * 7FH	: ***** 00	010	00 0000
B230	41	479	DB * 41H	: ***** 00	010	00 0000
B231	08	480	DB * 08H	: ***** 00	010	00 0000
		481				
B232	20	482	DB * 20H	: ***** 00	010	00 0000
B233	40	483	DB * 40H	: ***** 00	010	00 0000
B234	40	484	DB * 40H	: ***** 00	010	00 0000
B235	40	485	DB * 40H	: ***** 00	010	00 0000
B236	3F	486	DB * 3FH	: ***** 00	010	00 0000
		487				
B237	7F	488	DB * 7FH	: ***** 00	010	00 0000
B238	08	489	DB * 08H	: ***** 00	010	00 0000
B239	14	490	DB * 14H	: ***** 00	010	00 0000
B23A	22	491	DB * 22H	: ***** 00	010	00 0000
B23B	41	492	DB * 41H	: ***** 00	010	00 0000
		493				
B23C	7F	494	DB **** 7FH	: ***** 00	010	00 0000
B23D	40	495	DB * 40H	: ***** 00	010	00 0000
B23E	40	496	DB * 40H	: ***** 00	010	00 0000
B23F	40	497	DB * 40H	: ***** 00	010	00 0000
B240	40	498	DB **** 40H	: ***** 00	010	00 0000
		499				
B241	7F	500	DB **** 7FH	: ***** 00	010	00 0000
B242	02	501	DB * 02H	: ***** 00	010	00 0000
B243	0C	502	DB * 0CH	: ***** 00	010	00 0000
B244	02	503	DB * 02H	: ***** 00	010	00 0000
B245	7F	504	DB **** 7FH	: ***** 00	010	00 0000
		505				
B246	7F	506	DB **** 7FH	: ***** 00	010	00 0000
B247	04	507	DB * 04H	: ***** 00	010	00 0000
B248	08	508	DB * 08H	: ***** 00	010	00 0000
B249	10	509	DB * 10H	: ***** 00	010	00 0000
B24A	7F	510	DB **** 7FH	: ***** 00	010	00 0000
		511	511 \$EJECT			

024E 41	516	DB	41H	1	*	*	80	800	28	0550
024F 3E	517	DB	3EH	1	*	*	80	100	28	1550
	518			1	*****		80	500	10	5550
0250 7F	519	DB	7FH	1	*****		80	500	30	0550
0251 09	520	DB	09H	1	H10	*	80	200	10	4550
0252 09	521	DB	09H	1	H10	*	80	200	10	5550
0253 09	522	DB	09H	1	H10	*	80	500	10	4550
0254 06	523	DB	06H	1	H10	*	80	500	17	5550
	524							200		
0255 3E	525	DB	3EH	1	*****		80	500	37	0550
0256 41	526	DB	41H	1	*	*	80	100	08	0550
0257 51	527	DB	51H	1	*	*	80	500	08	4550
0258 21	528	DB	21H	1	H10	*	80	500	08	0550
0259 5E	529	DB	5EH	1	*	*****	80	400	37	5550
	530							200		
025A 7F	531	DB	7FH	1	*****		80	300	08	0550
025B 09	532	DB	09H	1	H10	*	80	500	10	5550
025C 19	533	DB	19H	1	H10	*	80	500	10	5550
025D 29	534	DB	29H	1	H10	*	80	500	10	0550
025E 46	535	DB	46H	1	H10	*	80	500	08	1050
	536							100		
025F 26	537	DB	26H	1	H10	*	80	500	02	0550
0260 49	538	DB	49H	1	H10	*	80	500	00	0550
0261 49	539	DB	49H	1	H10	*	80	500	00	4550
0262 49	540	DB	49H	1	H10	*	80	500	00	0550
0263 32	541	DB	32H	1	H10	*	80	500	30	0550
	542							500		
0264 01	543	DB	01H	1	H10	*	80	500	30	1050
0265 01	544	DB	01H	1	H10	*	80	500	00	0550
0266 7F	545	DB	7FH	1	*****		80	500	01	0550
0267 01	546	DB	01H	1	H10	*	80	100	55	0550
0268 01	547	DB	01H	1	H10	*	80	500	10	0550
	548							500		
0269 3F	549	DB	3FH	1	*****		80	500	30	0550
026A 40	550	DB	40H	1	H10	*	80	500	00	0550
026B 40	551	DB	40H	1	H10	*	80	500	00	0550
026C 40	552	DB	40H	1	H10	*	80	500	00	4550
026D 3F	553	DB	3FH	1	*****		80	500	00	0550
	554							200		
026E 1F	555	DB	1FH	1	H10	*	80	500	30	1050
026F 20	556	DB	20H	1	H10	*	80	100	50	0550
0270 40	557	DB	40H	1	H10	*	80	500	50	0550
0271 20	558	DB	20H	1	H10	*	80	500	50	0550
0272 1F	559	DB	1FH	1	H10	*	80	500	30	0550
	560							500		
0273 7F	561	DB	7FH	1	*****		80	500	30	0550
0274 20	562	DB	20H	1	H10	*	80	500	30	0550
0275 10	563	DB	10H	1	H10	*	80	500	00	0550
0276 20	564	DB	20H	1	H10	*	80	500	00	0550
0277 7F	565	DB	7FH	1	*****		80	500	30	0550
	566	#EJECT						500		

T0343M 110

LOC	OBJ	SEQ	SOURCE STATEMENT	THEN STATE	30000	300
		567			410	
B27B	63	568	DB 63H	MOV A, 63H	710	0000 0000
B279	14	569	DB 14H	MOV A, 14H	810	00 0000
B27A	08	570	DB 08H	MOV A, 08H	910	00 0000
B27B	14	571	DB 14H	MOV A, 14H	0020	0000 0000
B27C	63	572	DB 63H	MOV A, 63H	100	00 0000
		573		MOV A, 00H	200	00 0000
B27D	03	574	DB 03H	MOV A, 03H	300	00 0000
B27E	04	575	DB 04H	MOV A, 04H	400	00 0000
B27F	78	576	DB 78H	MOV A, 78H	500	00 0000
B280	04	577	DB 04H	MOV A, 04H	600	00 0000
B281	03	578	DB 03H	MOV A, 03H	700	00 0000
		579		MOV A, 00H	800	00 0000
B282	61	580	DB 61H	MOV A, 61H	900	00 0000
B283	51	581	DB 51H	MOV A, 51H	0000	00 0000
B284	49	582	DB 49H	MOV A, 49H	100	00 0000
B285	45	583	DB 45H	MOV A, 45H	200	00 0000
B286	43	584	DB 43H	MOV A, 43H	300	00 0000
		585		MOV A, 00H	400	00 0000
B287	7F	586	DB 7FH	MOV A, 7FH	500	00 0000
B288	7F	587	DB 7FH	MOV A, 7FH	600	00 0000
B289	41	588	DB 41H	MOV A, 41H	700	00 0000
B28A	41	589	DB 41H	MOV A, 41H	800	00 0000
B28B	41	590	DB 41H	MOV A, 41H	900	00 0000
		591		MOV A, 00H	0000	00 0000
B28C	02	592	DB 02H	MOV A, 02H	100	0000 0000
B28D	04	593	DB 04H	MOV A, 04H	200	0000 0000
B28E	08	594	DB 08H	MOV A, 08H	300	0000 0000
B28F	10	595	DB 10H	MOV A, 10H	400	0000 0000
B290	20	596	DB 20H	MOV A, 20H	500	0000 0000
		597		MOV A, 00H	600	0000 0000
B291	41	598	DB 41H	MOV A, 41H	700	0000 0000
B292	41	599	DB 41H	MOV A, 41H	800	0000 0000
B293	41	600	DB 41H	MOV A, 41H	900	0000 0000
B294	7F	601	DB 7FH	MOV A, 7FH	0000	0000 0000
B295	7F	602	DB 7FH	MOV A, 7FH	1000	0000 0000
		603		MOV A, 00H	2000	0000 0000
B296	10	604	DB 10H	MOV A, 10H	3000	0000 0000
B297	08	605	DB 08H	MOV A, 08H	4000	0000 0000
B298	04	606	DB 04H	MOV A, 04H	5000	0000 0000
B299	08	607	DB 08H	MOV A, 08H	6000	0000 0000
B29A	10	608	DB 10H	MOV A, 10H	7000	0000 0000
		609		MOV A, 00H	8000	0000 0000
B29B	40	610	DB 40H	MOV A, 40H	9000	0000 0000
B29C	40	611	DB 40H	MOV A, 40H	0000	0000 0000
B29D	40	612	DB 40H	MOV A, 40H	1000	0000 0000
B29E	40	613	DB 40H	MOV A, 40H	2000	0000 0000
B29F	40	614	DB 40H	MOV A, 40H	3000	0000 0000
		615	*EJECT			

LOC	OBJ	SEQ	SOURCE STATEMENT	INSTR	STATUS	PC	PC+1
		616	:				
02A0	0800	617	PAGE1: MOV STBCNT, #00H ;ZERO STROBE COUNTER	MOV		0000	0000
02A2	FA	618	MOV A, SAYPNT ;GET DIRECTION	MOV		0000	0000
02A3	37	619	CPL A ;FLIP BITS	CPL		0000	0000
02A4	D2B3	620	JB6 BAKWRD ;IF BACKWARD JUMP OUT	JB		0000	0000
02A6	FC	621	LKLO: MOV A, TEMP1 ;GET THE TARGET	MOV		0000	0000
02A7	A3	622	MOVP A, @A ;GET THE DATA	MOVP		0000	0000
02A8	34CC	623	CALL FIRE ;STROBE THE SOLENOIDS	CALL		0000	0000
02AA	1C	624	INC TEMP1 ;INCREMENT THE POINTER	INC		0000	0000
02AB	1B	625	INC STBCNT ;INCREMENT THE STROBE COUNTER	INC		0000	0000
02AC	FB	626	MOV A, STBCNT ;GET THE STROBE COUNTER	MOV		0000	0000
02AD	D3B5	627	XRL A, #05H ;IS IT FIVE	XRL		0000	0000
02AF	96A6	628	JNZ LKLO ;REPEAT IF NOT FIVE	JNZ		0000	0000
02B1	84AE	629	JMP SETTIM ;GO BACK	JMP		0000	0000
02B3	FC	630	BAKWRD: MOV A, TEMP1 ;GET THE TARGET	MOV		0000	0000
02B4	03B4	631	ADD A, #04H ;COMPENSATE FOR GOING BACKWARDS	ADD		0000	0000
02B6	AC	632	MOV TEMP1, A ;SAVE IT	MOV		0000	0000
02B7	FC	633	LKLO1: MOV A, TEMP1 ;GET THE TARGET	MOV		0000	0000
02B8	A3	634	MOVP A, @A ;GET THE DATA	MOVP		0000	0000
02B9	34CC	635	CALL FIRE ;STROBE THE SOLENOIDS	CALL		0000	0000
02BB	FC	636	MOV A, TEMP1 ;GET TEMP1	MOV		0000	0000
02BC	07	637	DEC A ;DECREASE BY ONE	DEC		0000	0000
02BD	AC	638	MOV TEMP1, A ;PUT IT BACK	MOV		0000	0000
02BE	1B	639	INC STBCNT ;INCREMENT THE STROBE COUNTER	INC		0000	0000
02BF	FB	640	MOV A, STBCNT ;GET THE STROBE COUNTER	MOV		0000	0000
02CB	D3B5	641	XRL A, #05H ;IS IT FIVE	XRL		0000	0000
02C2	96B7	642	JNZ LKLO1 ;REPEAT IF NOT FIVE	JNZ		0000	0000
02C4	84AE	643	JMP SETTIM ;GO BACK, CHARACTER IS DONE	JMP		0000	0000
		644	\$EJECT				

LDC	OBJ	SEQ	SOURCE STATEMENT	THREATAT2	33R002	033	400	301
		645	;*			1P3		
0300		646	ORG 300H	H00	00	0P3	00	0000
		647	;*	H00	00	0P3	00	0000
		648		H00	00	0P3	00	0000
0300 00		649	DB 00H	H00	00	0P3	00	0000
0301 00		650	DB 00H	H00	00	0P3	00	0000
0302 00		651	DB 00H	H00	00	0P3	00	0000
0303 00		652	DB 00H	H00	00	0P3	00	0000
0304 00		653	DB 00H	H00	00	0P3	00	0000
		654		H00	00	0P3	00	0000
0305 00		655	DB 00H	H00	00	0P3	00	0000
0306 00		656	DB 00H	H00	00	0P3	00	0000
0307 5F		657	DB 5FH	H00	00	0P3	00	0000
0308 00		658	DB 00H	H00	00	0P3	00	0000
0309 00		659	DB 00H	H00	00	0P3	00	0000
		660		H00	00	0P3	00	0000
030A 00		661	DB 00H	H00	00	0P3	00	0000
030B 07		662	DB 07H	H00	00	0P3	00	0000
030C 00		663	DB 00H	H00	00	0P3	00	0000
030D 07		664	DB 07H	H00	00	0P3	00	0000
030E 00		665	DB 00H	H00	00	0P3	00	0000
		666		H00	00	0P3	00	0000
030F 14		667	DB 14H	H00	00	0P3	00	0000
0310 7F		668	DB 7FH	H00	00	0P3	00	0000
0311 14		669	DB 14H	H00	00	0P3	00	0000
0312 7F		670	DB 7FH	H00	00	0P3	00	0000
0313 14		671	DB 14H	H00	00	0P3	00	0000
		672		H00	00	0P3	00	0000
0314 24		673	DB 24H	H00	00	0P3	00	0000
0315 2A		674	DB 2AH	H00	00	0P3	00	0000
0316 7F		675	DB 7FH	H00	00	0P3	00	0000
0317 2A		676	DB 2AH	H00	00	0P3	00	0000
0318 12		677	DB 12H	H00	00	0P3	00	0000
		678		H00	00	0P3	00	0000
0319 23		679	DB 23H	H00	00	0P3	00	0000
031A 13		680	DB 13H	H00	00	0P3	00	0000
031B 00		681	DB 00H	H00	00	0P3	00	0000
031C 64		682	DB 64H	H00	00	0P3	00	0000
031D 62		683	DB 62H	H00	00	0P3	00	0000
		684		H00	00	0P3	00	0000
031E 36		685	DB 36H	H00	00	0P3	00	0000
031F 49		686	DB 49H	H00	00	0P3	00	0000
0320 56		687	DB 56H	H00	00	0P3	00	0000
0321 20		688	DB 20H	H00	00	0P3	00	0000
0322 50		689	DB 50H	H00	00	0P3	00	0000
		690	#EJECT	H00	00	0P3	00	0000

0324 00	693	DB	00H	/			
0325 07	694	DB	07H	/	***		
0326 00	695	DB	00H	/	000	00	
0327 00	696	DB	00H	/	000	00	
	697			/	000	00	
0328 1C	698	DB	1CH	/	000	***	00
0329 22	699	DB	22H	/	000	* *	00
032A 41	700	DB	41H	/	*		*
032B 00	701	DB	00H	/	000	00	
032C 00	702	DB	00H	/	000	00	
	703			/	000	00	
032D 00	704	DB	00H	/	000	00	
032E 00	705	DB	00H	/	000	00	
032F 41	706	DB	41H	/	*		*
0330 22	707	DB	22H	/	000	* *	00
0331 1C	708	DB	1CH	/	000	***	00
	709			/	000		00
0332 22	710	DB	22H	/	000	* *	00
0333 14	711	DB	14H	/	000	* *	00
0334 7F	712	DB	7FH	/	*****		
0335 14	713	DB	14H	/	000	* *	00
0336 22	714	DB	22H	/	000	* *	00
	715			/	000		00
0337 00	716	DB	00H	/	000	* *	00
0338 00	717	DB	00H	/	000	* *	00
0339 7F	718	DB	7FH	/	*****		
033A 00	719	DB	00H	/	000	* *	00
033B 00	720	DB	00H	/	000	* *	00
	721			/	000		00
033C 00	722	DB	00H	/	000	* *	00
033D 40	723	DB	40H	/	000	* *	00
033E 30	724	DB	30H	/			
033F 00	725	DB	00H	/	000		
0340 00	726	DB	00H	/	000		
	727			/	000		
0341 00	728	DB	00H	/	000	* *	00
0342 00	729	DB	00H	/	000	* *	00
0343 00	730	DB	00H	/			
0344 00	731	DB	00H	/	000	* *	00
0345 00	732	DB	00H	/	000	* *	00
	733			/	000		
0346 00	734	DB	00H	/	000		
0347 00	735	DB	00H	/	000		
0348 40	736	DB	40H	/			
0349 00	737	DB	00H	/	*		
034A 00	738	DB	00H	/			
	739			/			
034B 20	740	DB	20H	/	*		
034C 10	741	DB	10H	/	*		
034D 00	742	DB	00H	/	*		
034E 04	743	DB	04H	/	*		
034F 02	744	DB	02H	/	*		
	745			/			
0350 3E	746	DB	3EH	/	*****		
0351 51	747	DB	51H	/	* * *		
0352 49	748	DB	49H	/	* * *		
0353 45	749	DB	45H	/	* * *		
0354 3E	750	DB	3EH	/	*****		
	751			/			
0355 00	752	DB	00H	/			
0356 42	753	DB	42H	/	* * *		
0357 7F	754	DB	7FH	/	*****		
0358 40	755	DB	40H	/	*		
0359 00	756	DB	00H	/			
	757			/			
035A 62	758	DB	62H	/	* * *		
035B 51	759	DB	51H	/	* * *		
035C 49	760	DB	49H	/	* * *		
035D 49	761	DB	49H	/	* * *		
035E 46	762	DB	46H	/	* * *		
	763			/			
035F 21	764	DB	21H	/	* * *		
0360 41	765	DB	41H	/	* * *		

LOC	OBJ	SEQ	SOURCE STATEMENT	THREMTNT2 B1R002	002	000	001
		000					
0370	46	001	DB 46H				
037E	49	002	DB 49H				
037F	49	003	DB 49H				
0380	29	004	DB 29H				
0381	1E	005	DB 1EH				
		006					
0382	00	007	DB 00H				
0383	00	008	DB 00H				
0384	14	009	DB 14H				
0385	00	010	DB 00H				
0386	00	011	DB 00H				
		012					
0387	00	013	DB 00H				
0388	40	014	DB 40H				
0389	34	015	DB 34H				
038A	00	016	DB 00H				
038B	00	017	DB 00H				
		018					
038C	00	019	DB 00H				
038D	14	020	DB 14H				
038E	22	021	DB 22H				
038F	41	022	DB 41H				
0390	00	023	DB 00H				
		024					
0391	14	025	DB 14H				
0392	14	026	DB 14H				
0393	14	027	DB 14H				
0394	14	028	DB 14H				
0395	14	029	DB 14H				
		030					
0396	00	031	DB 00H				
0397	41	032	DB 41H				
0398	22	033	DB 22H				
0399	14	034	DB 14H				
039A	00	035	DB 00H				
		036					
039B	02	037	DB 02H				
039C	01	038	DB 01H				
039D	59	039	DB 59H				
039E	05	040	DB 05H				
039F	02	041	DB 02H				
		042	*EJECT				

LOC	OBJ	SEQ	SOURCE STATEMENT	INSTR	COND	LOC	OBJ
B3A8	B88B	843	PAGE2: MOV STBCNT, #8BH ;ZERO STROBE COUNTER	MOV			
B3A2	FA	844	MOV A, SAYPHT ;GET DIRECTION	MOV			
B3A3	37	845	CPL A ;FLIP BITS	CPL			
B3A4	D2B5	846	JBC BKWRD ;IF BACKWARD JUMP OUT	JBC			
B3A6	FC	847	LKHI: MOV A, TEMP1 ;GET THE TARGET	MOV			
B3A7	B36B	848	ADD A, #6BH ;ADJUST THE TARGET	ADD			
B3A9	A3	849	MOV A, @A ;GET THE DATA	MOV			
B3AA	34CC	850	CALL FIRE ;STROBE THE SOLENOIDS	CALL			
B3AC	1C	851	INC TEMP1 ;INCREMENT THE POINTER	INC			
B3AD	1B	852	INC STBCNT ;INCREMENT THE STROBE COUNTER	INC			
B3AE	FB	853	MOV A, STBCNT ;GET THE STROBE COUNTER	MOV			
B3AF	D3B5	854	XRL A, #B5H ;IS IT FIVE	XRL			
B3B1	96A6	855	JNZ LKHI: ;REPEAT IF NOT FIVE	JNZ			
B3B3	84AE	856	JMP SETTIM ;GO BACK	JMP			
B3B5	FC	857	MOV A, TEMP1 ;GET THE TARGET	MOV			
B3B6	B364	858	ADD A, #64H ;COMPENSATE FOR GOING BACKWARDS	ADD			
B3B8	AC	859	MOV TEMP1, A ;SAVE IT	MOV			
B3B9	FC	860	MOV A, TEMP1 ;GET THE TARGET	MOV			
B3BA	A3	861	MOV A, @A ;GET THE DATA	MOV			
B3BB	34CC	862	CALL FIRE ;STROBE THE SOLENOIDS	CALL			
B3BD	FC	863	MOV A, TEMP1 ;GET TEMP1	MOV			
B3BE	B7	864	DEC A ;DECREASE BY ONE	DEC			
B3BF	AC	865	MOV TEMP1, A ;PUT IT BACK	MOV			
B3C0	1B	866	INC STBCNT ;INCREMENT THE STROBE COUNTER	INC			
B3C1	FB	867	MOV A, STBCNT ;GET THE STROBE COUNTER	MOV			
B3C2	D3B5	868	XRL A, #B5H ;IS IT FIVE	XRL			
B3C4	96B9	869	JNZ LKHI: ;REPEAT IF NOT FIVE	JNZ			
B3C6	84AE	870	JMP SETTIM ;GO BACK, CHARACTER IS DONE	JMP			
		871	*EJECT				

8400 27	874	; STIS 9134							
8401 90	875	BCIN: CLR	WNOA	A					
8402 94B8	876	MOVX	INT	PRDA					
8404 943F	877	CALL	Y0	SETUP					
8406 84BA	878	CALL	INT	VARSET					
	879	JMP	PRNT						
	880								
8408 23FE	881	SETUP:	MOV	WNOA	A	#BFEH			
840A 39	882	OUTL	INT	P1	A				
	883								
	884	MOV	DELAY	3.2 SECONDS					
	885								
840B BC05	886	MOV	INT	TEMP1	#05H				
840D BFFF	887	SELFC:	MOV	JUNK1	#BFFH				
840F BEFF	888	SELFB:	MOV	TI	LINCNT	#BFFH			
8411 B9	889	SELFA:	IN	A	P1				
8412 37	890	CPL	INT	A					
8413 F21D	891	JB7	INT	06	DONE				
8415 EE11	892	DJNZ	LINCNT	SELFA					
8417 EFBF	893	DJNZ	JUNK1	SELFB					
8419 EC0D	894	DJNZ	TI	TEMP1	SELFC				
841B 845A	895	JMP	ERROR						
	896								
	897								
	898								
841D BFFF	899	DONE:	MOV	JUNK1	#BFFH				
841F BEFF	900	SELF:	MOV	LINCNT	#BFFH				
8421 B9	901	SELF1:	IN	A	P1				
8422 F22A	902	JB7	DONE						
8424 EE21	903	DJNZ	LINCNT	SELF1					
8426 EF1F	904	DJNZ	JUNK1	SELF					
8428 845A	905	JMP	ERROR						
	906								
	907								
	908								
	909								
842A BC04	910	DONE:	MOV	TEMP1	#04H				
842C BFFF	911	SELFCC:	MOV	JUNK1	#BFFH				
842E BEFF	912	SELFB:	MOV	LINCNT	#BFFH				
8430 B9	913	SELFA:	IN	A	P1				
8431 37	914	CPL	A						
8432 D23C	915	JB6	DONE						
8434 EE3B	916	DJNZ	LINCNT	SELFA					
8436 EF2E	917	DJNZ	JUNK1	SELFB					
8438 EC2C	918	DJNZ	TEMP1	SELFCC					
843A 845A	919	JMP	ERROR						
843C 89B1	920	DONE:	ORL	P1	#01H				
843E 83	921	RET							
	922								
	923								
	924								
843F 23FE	925	VARSET:	MOV	A	#BFEH				
8441 62	926		MOV	T	A				
8442 55	927	STRT	T						
8443 B82B	928	MOV	INBUF	#FIRST					
8445 BE0B	929	MOV	LINCNT	#0BH					
8447 BD0B	930	MOV	STATUS	#0BH					
	931								
	932								
	933								
8449 B92B	934	MOV	OUTBUF	#FIRST					
844B 232B	935	CLRMEM:	MOV	A	#2BH				
844D A1	936		MOV	OUTBUF	A				
844E 19	937	INC	OUTBUF						
844F F9	938	MOV	A	OUTBUF					
845B D37B	939	XRL	A	#MAX+1					
8452 964B	940	JNZ	CLRMEM						
	941								
	942								
	943								
8454 99EF	944	ANL	P1	#BEFH					
8456 8B	945	MOVX	A	INBUF					
8457 891B	946	ORL	P1	#1BH					
	947								

LOC	OBJ	SEQ	SOURCE STATEMENT	THE STATE	338002	032	400	301
		948	HOWEXIT: VARSET					
		949						
B459	83	950	RET					
		951						
		952	THIS ROUTINE TURNS THE MOTOR OFF AND LOOPS					
		953						
B45A	89FF	954	ERROR: ORL P1, #BFFH					
B45C	845C	955	DEAD: JMP DEAD					
		956						
		957	THESE ARE ALL SUBROUTINES THAT ARE CALLED					
		958						
B45E	19	959	INCTST: INC OUTBUF					
B45F	237B	960	MOV A, #MAX+1					
B461	D9	961	XRL A, OUTBUF					
B462	83	962	RET					
B463	B9	963	GTPRNT: IN A, P1					
B464	37	964	CPL A					
B465	D263	965	JB6 GTPRNT					
B467	166B	966	TSTJTF: JTF					
B469	8467	967	JMP TSTJTF					
B46B	65	968	STOP TCNT					
B46C	FF	969	MOV A, JUNK1					
B46D	34C1	970	CALL PRNTIT					
B46F	341C	971	CALL LMODE					
B471	83	972	RET					
B472	F9	973	DECTST: MOV A, OUTBUF					
B473	B7	974	DEC A					
B474	A9	975	MOV OUTBUF, A					
B475	D31F	976	XRL A, #FIRST-1					
B477	83	977	RET					
		978						
		979	THIS ROUTINE DOES A LINE FEED					
		980						
B478	FE	981	LINEFD: MOV A, LINCNT					
B479	F29B	982	JB7 DOFF					
B47B	99FD	983	LFDD: P1, #BFDH					
B47D	BC4D	984	MOV TEMP1, #40H					
B47F	BF93	985	MOV JUNK1, #93H					
B481	EF81	986	LFLP2: DJNZ JUNK1, LFLP2					
B483	EC7F	987	DJNZ TEMP1, LFLP1					
B485	89B2	988	ORL P1, #B2H					
B487	1E	989	INC LINCNT					
B488	FE	990	MOV A, LINCNT					
B489	D32B	991	XRL A, #20H					
B48B	96BF	992	JNZ NOTDON					
B48D	BE8B	993	MOV LINCNT, #BBH					
		994						
		995	NOW DELAY 98 MILLISEC					
		996						
B48F	BC8B	997	NOTDON: MOV TEMP1, #8BH					
B491	BFFF	998	LOP1: MOV JUNK1, #BFFH					
B493	EF93	999	LOP2: DJNZ JUNK1, LOP2					
B495	EC91	1000	DJNZ TEMP1, LOP1					
B497	83	1001	RET					
		1002						
		1003	THIS ROUTINE DOES A FORM FEED					
		1004						
B498	B9	1005	DOFF: IN A, P1					
B499	37	1006	CPL A					
B49A	53C8	1007	ANL A, #BCBH					
B49C	C69B	1008	JZ DOFF					
B49E	89B1	1009	ORL P1, #B1H					
B4A0	947B	1010	CALL LFDD					
B4A2	FE	1011	FFCK: MOV A, LINCNT					
B4A3	537F	1012	ANL A, #7FH					
B4A5	D3BB	1013	XRL A, #BBH					
B4A7	C6AD	1014	JZ					
B4A9	947B	1015	CALL LFDD					
B4AB	84A2	1016	JMP FFCK					
B4AD	83	1017	FFDONE: RET					
		1018						
B4AE	23EB	1019	SETTIM: MOV A, #BEBH					
B4B0	62	1020	MOV T, A					
B4B1	55	1021	STRT T					
B4B2	83	1022	RET					
		1023						

LDC	OBJ	SEQ	SOURCE STATEMENT
B4B3	42	1024	PRNTBK: MOV A,T ;GET THE TIMER
B4B4	37	1025	CPL A ;TWO'S COMPLEMENT ACC
B4B5	17	1026	INC A
B4B6	17	1027	INC A
B4B7	17	1028	INC A
B4B8	17	1029	INC A
B4B9	17	1030	INC A ;ADJUST TIMER
B4BA	62	1031	MOV T,A ;PUT IT BACK IN THE TIMER
B4BB	09	1032	INLOOP: IN A,P1 ;READ PORT 1
B4BC	F2C8	1033	J87 CONPBK ;IF SENSOR IN NOT COVERED, LEAVE
B4BE	0488	1034	JMP INLOOP ;OTHERWISE LOOP
B4C0	55	1035	CONPBK: STRT T ;START THE TIMER
B4C1	16C5	1036	CONPB: JTF RDOPT ;SEE IF READY TO PRINT
B4C3	84C1	1037	JMP CONPB ;OTHERWISE LOOP
B4C5	23FF	1038	RDOPT: MOV A,#BFFH ;LOAD A
B4C7	62	1039	MOV T,A ;PUT IT IN THE TIMER
B4C8	83	1040	RET ;EXIT
		1041	;
		1042	;THIS ROUTINE ADJUSTS AND SAVES THE STATUS DURING PRINTING
		1043	;
B4C9	FD	1044	STACHK: MOV A,STATUS ;GET THE STATUS
B4CA	92D2	1045	LFSET JB4 ;SET LINE FEED BIT
B4CC	AA	1046	B4RET: MOV SAVPNT,A ;SAVE THE STATUS
B4CD	53C2	1047	ANL A,#BC2H ;RESET EVERYTHING EXCEPT
		1048	;
B4CF	AD	1049	MOV STATUS,A ;DIRECTION AND PRINT
B4DB	0413	1050	JMP LPRNT1 ;PUT THE STATUS BACK
B4D2	432B	1051	LFSET: ORL A,#2BH ;EXIT
B4D4	84CC	1052	JMP B4RET ;SET BIT 5
		1053	;
		1054	;THIS ROUTINE READS A CHARACTER AND PUTS IT IN THE ACC
		1055	;
B4D6	99EF	1056	GTCHAR: ANL P1,#BEFH ;SET ENABLE BIT
B4D8	88	1057	MOVX A,@INBUF ;READ THE CHARACTER
B4D9	891B	1058	ORL P1,#1BH ;RESET ENABLE BIT
B4DB	83	1059	RET ;EXIT GTCHAR
		1060	;
		1061	;THIS ROUTINE TURNS THE MOTOR ON
		1062	;
B4DC	99FE	1063	MDTON: ANL P1,#BFEH ;TURN MOTOR ON
B4DE	83	1064	RET ;EXIT
		1065	;
		1066	;THIS ROUTINE TURNS THE MOTOR OFF
		1067	;
B4DF	89B1	1068	MDTOF: ORL P1,#B1H ;TURN MOTOR OFF
B4E1	83	1069	RET ;EXIT
		1070	;
		1071	END ;DONE
USER SYMBOLS			
ARND	B187	ARNDJP	B149
CASE0	0031	CASE01	0017
CONPB	B4C1	CONPBK	B4C8
DOLF	0071	DONEF	042A
FDCR1	00A8	FFCK	04A2
FIRST	0020	FIXDUM	0174
GTCHAR	0406	GTPRNT	0463
LDBUF	0108	LFCKCK	014A
LINDNT	0086	LINEFD	0478
LOP1	0491	LOP2	0493
MOLF	011A	MDTON	040F
PIT	0468	PRNT	008A
SELFA	0411	SELFAA	0430
SHORT	01CA	STACHK	04C9
TEMP1	00B4	TSJTF	01DC
B4RET	B4CC	BAKWRD	B2R3
CASE1	0052	CASE2	0080
CRFIX	01A8	CRFND	00D2
DONEL	043C	DONER	0410
FFDNE	04A0	FFFIX	0182
FIXFIN	018F	FIXUP	0189
INBUF	0088	INCTST	045E
LFCKCK	014A	LFCKCK	014A
LKHI	03A6	LKHI1	0389
LPRNT	0011	LPRNT1	0013
HT1	0104	OUTBUF	00B1
PRNTBK	04B3	PRNTIT	01C1
SELFB	040F	SELFB8	042E
STATUS	0085	STBCNT	0083
TSJTF	0467	VARSET	043F
BCIN	0480	BKWRD	03B5
CASE23	0024	CASE3	00C2
CRFND	00D2	DEAD	045C
ERROR	045A	FDC	0042
FINE	0178	FIRE	01CC
FXCHAR	0161	FXPRNT	0191
INLOOP	04B8	ISCHAR	0180
LFCKCK	014A	LFCKCK	014A
LKLO1	02A6	LKLO1	02B7
MAX	006F	MDTOF	040F
OVR	008A	OVR1	0085
RDOPT	04C5	SAVPNT	0082
SELFC	0480	SELFC	042C
STBTT1	0158	STPRNT	0159
WATCH	0075	WATCHD	00AE
BUTLOP	B113	BYEYE	B168
CHAR	B11F	CLRMEM	B448
DECTST	B472	DOFF	B498
FDC1	B044	FDCR	B09E
FIREX	B108	FIREY	B106
GETSTA	B144	G000	B128
JUNK1	B0B7	KTDUM	B1E0
LFTEST	B17F	LFTEST	B17F
LNRDDE	B11C	LOOPW	B074
MOTON	B40C	NOFF	B18F
PAGE1	B2AB	PAGE2	B3AB
SELF	B41F	SELF1	B421
SETTIM	B44E	SETUP	B488
SUB1	B139	TAB1E1	B2B0
ASSEMBLY COMPLETE. NO ERRORS			

Microcontroller includes a-d converter for lowest-cost analog interfacing

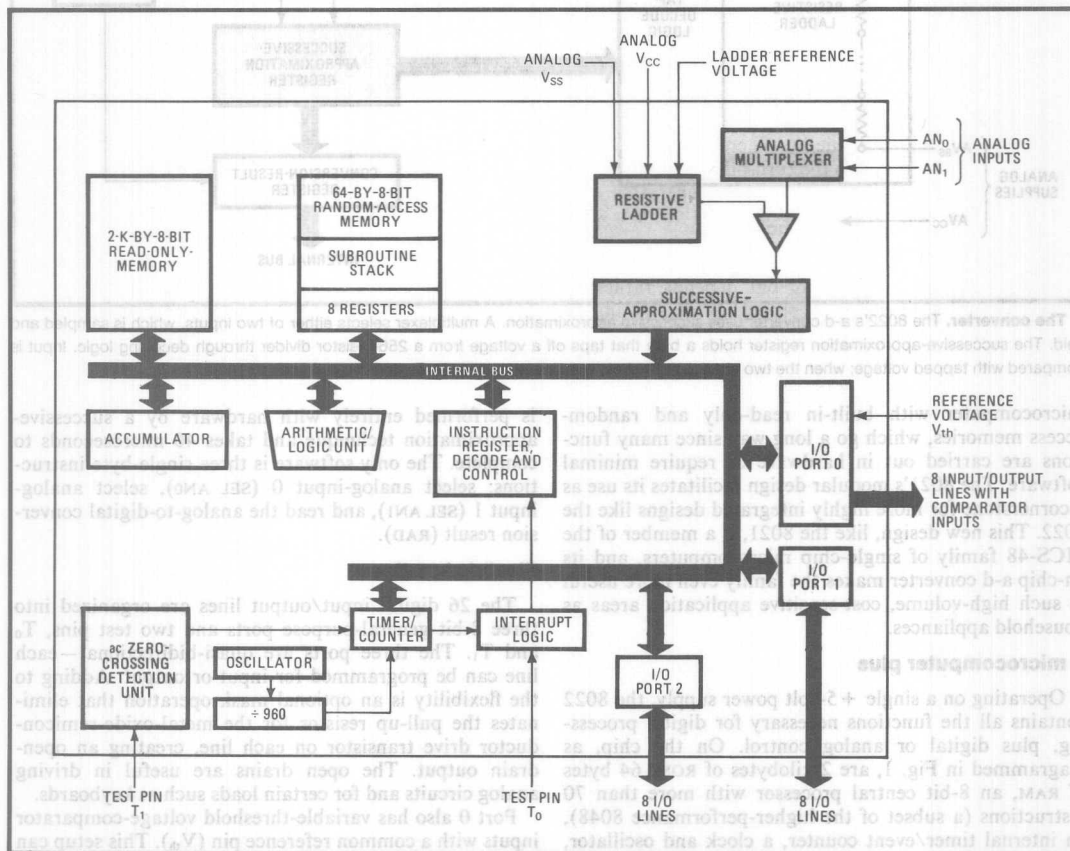
Adding hardware for analog-to-digital conversion to a single-chip microcomputer cuts interface software and component count for high-volume control applications

by W. Check, E. Cheng, G. Hill, M. Hollen, and J. Miller, Intel Corp., Santa Clara, Calif.

Microcomputers' plunging size and cost are creating a rising new market: low-cost controllers that end up in automobiles, appliances, and consumer products. Now that the technology is available to integrate a high-performance 8-bit analog-to-digital converter and a microcomputer on a single chip, the tremendous need for

low-cost analog interfacing has hastened the development of just such a device: the 8022. By integrating the a-d converter and other useful features, the chip achieves the minimum system cost possible for high-volume controller applications involving analog signals.

The heart of the 8022 is the 8021 general-purpose

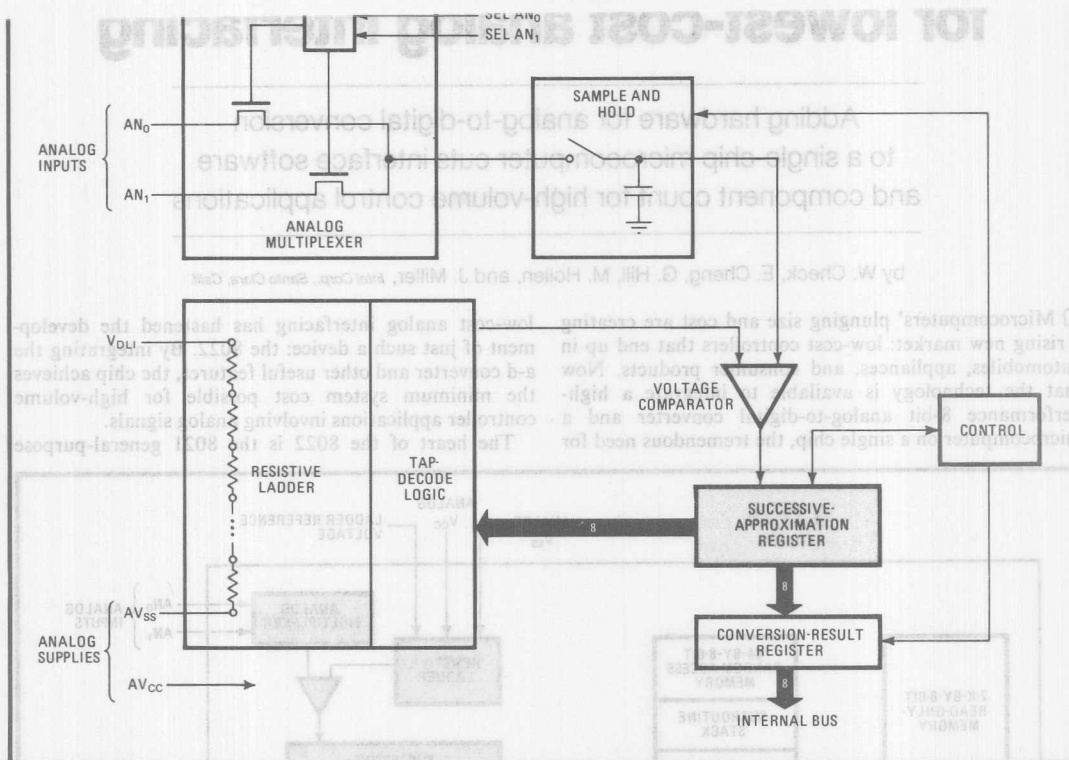


1. All aboard. The first single-chip microcomputer with a built-in 8-bit analog-to-digital converter is Intel's 8022. Around a foundation of the 8021, the chip packs several features that suit it to control applications: two multiplexed analog inputs, a zero-crossing detector, two 7-mA digital outputs that are part of Port 1, and a total of 26 digital input/output lines, eight of which have voltage-comparator inputs.

Reprinted from *Electronics*/ May 25, 1978

Copyright Cahners Publishing Co., Inc. 1977. All rights reserved.

Electronics/May 25, 1978



2. The converter. The 8022's a-d converter uses successive approximation. A multiplexer selects either of two inputs, which is sampled and held. The successive-approximation register holds a byte that taps off a voltage from a 256-resistor divider through decoding logic. Input is compared with tapped voltage; when the two are equal, the held byte is sent to the conversion-result register.

microcomputer with built-in read-only and random-access memories, which go a long way since many functions are carried out in hardware or require minimal software. The 8021's modular design facilitates its use as a cornerstone for more highly integrated designs like the 8022. This new design, like the 8021, is a member of the MCS-48 family of single-chip microcomputers, and its on-chip a-d converter makes the family even more useful in such high-volume, cost-sensitive application areas as household appliances.

A microcomputer plus

Operating on a single +5-volt power supply, the 8022 contains all the functions necessary for digital processing, plus digital or analog control. On the chip, as diagrammed in Fig. 1, are 2 kilobytes of ROM, 64 bytes of RAM, an 8-bit central processor with more than 70 instructions (a subset of the higher-performance 8048), an internal timer/event counter, a clock and oscillator, the 8-bit a-d converter with two analog inputs, and 26 digital input/output lines.

All parts of the a-d converter are integrated onto the chip—no external components are required. Conversion

is performed entirely with hardware by a successive-approximation technique and takes 40 microseconds to complete. The only software is three single-byte instructions: select analog-input 0 (SEL AN0), select analog-input 1 (SEL AN1), and read the analog-to-digital conversion result (RAD).

Flexible I/O lines

The 26 digital input/output lines are organized into three 8-bit general-purpose ports and two test pins, T_0 and T_1 . The three ports are quasi-bidirectional—each line can be programmed for input or output. Adding to the flexibility is an optional mask operation that eliminates the pull-up resistor for the metal-oxide-semiconductor drive transistor on each line, creating an open-drain output. The open drains are useful in driving analog circuits and for certain loads such as keyboards.

Port 0 also has variable-threshold voltage-comparator inputs with a common reference pin (V_{th}). This setup can accommodate such input situations as high noise margins, low-voltage (10-to-15-v) touch switching, and expansion of the analog inputs. Two input/output pins ($P1_0$ and $P1_1$) provide for high-current drive; each sinks

display formatting with just a single 2-byte instruction.

The 64-byte RAM integrates the hardware stack and data memory. The first eight memory locations are designated as working registers and are addressable by any of the 11 direct-register instructions. Besides increasing the variety of operations that can be performed on data in memory, this approach further reduces the number of instruction bytes required for processing. Working registers 0 and 1 also may be used as pointers to indirectly address all locations in memory, using the indirect-register instructions.

The next 16 bytes of RAM may be used as the address stack to enable the processor to keep track of the return addresses generated from call instructions and to handle interrupts. Since each address is 11 bits long, 2 bytes are needed to store each address. Thus, the 16 bytes of address stack allow a total of altogether eight levels of subroutine nesting.

A 3-bit stack pointer supplies the locations that are loaded with the next return address generated. This stack pointer is incremented when a return address is stored and decremented when an address is fetched during a return. If an application does not require all eight levels of subroutine nesting, the free portion of the address stack may be used as standard RAM.

Other on-chip features

The 8022 contains its own clock and oscillator circuitry and requires only an external timing control element to generate all internal timing signals. For highly cost-sensitive applications an inductor may be used as this element. If a more precise clock is required, the designer may specify a crystal or external clock for the application.

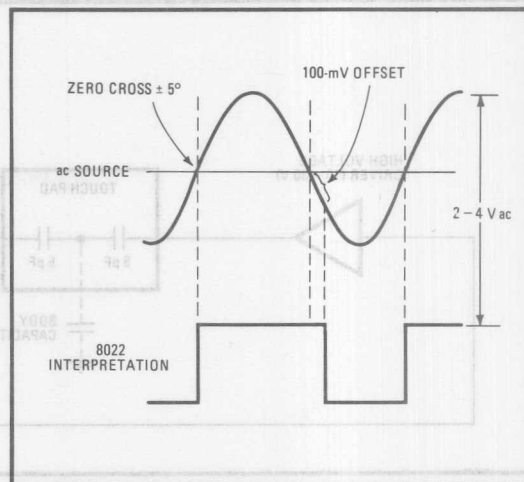
To further reduce the user's system cost and to permit use of the chip in noisy environments, the power-supply tolerance has been increased, permitting a range from 4.5 to 6.5 v. Less filtering and regulation is necessary, therefore, and the microcomputer's immunity to noisy power supplies is greater, as well.

The programmable 8-bit timer/event counter accurately monitors elapsed time, avoiding the software overhead of timing loops. Once it has been loaded with the contents of the accumulator, its divide-by-32 prescaler is incremented for each system clock cycle and at prescaler overflow. A timer flag is set at overflow. Once activated, it can be tested by a conditional-branch instruction to generate an interrupt. Total count capacity is 8,192 instruction cycles or 81.9 milliseconds, for the 10- μ s cycle time.

The timer may also be used as an event counter where the test pin T₁ serves as a counter input. Upon command, the chip will respond to a low-to-high transition on the pin by incrementing its timer.

Comparator inputs

The input/output port 0 of the 8022 has several properties that ease analog interfacing problems. Two of these features are moderate-gain voltage comparators and pull-up resistors on each line that either may serve as standard TTL outputs or may be masked out to give open-drain outputs.



4. Zero-crossing detector. Useful in timing the firing of triacs for ac phase control of appliances or getting a real-time clock, the 8022's T₁ test pin detects the crossing of a waveform's dc level by its rising edge. One hundred millivolts of hysteresis prevents chattering, and the ac frequency is limited to 1 kilohertz.

The comparators are especially handy for troublesome inputs. The comparator at each pin accurately compares that line to the threshold-voltage reference pin, V_{th}, within about 100 millivolts in the range from V_{ss} to V_{cc}/2. Allowed to float, V_{th} will bias itself to the digital switch point of the other ports, and port 0 then behaves as a set of conventional digital inputs.

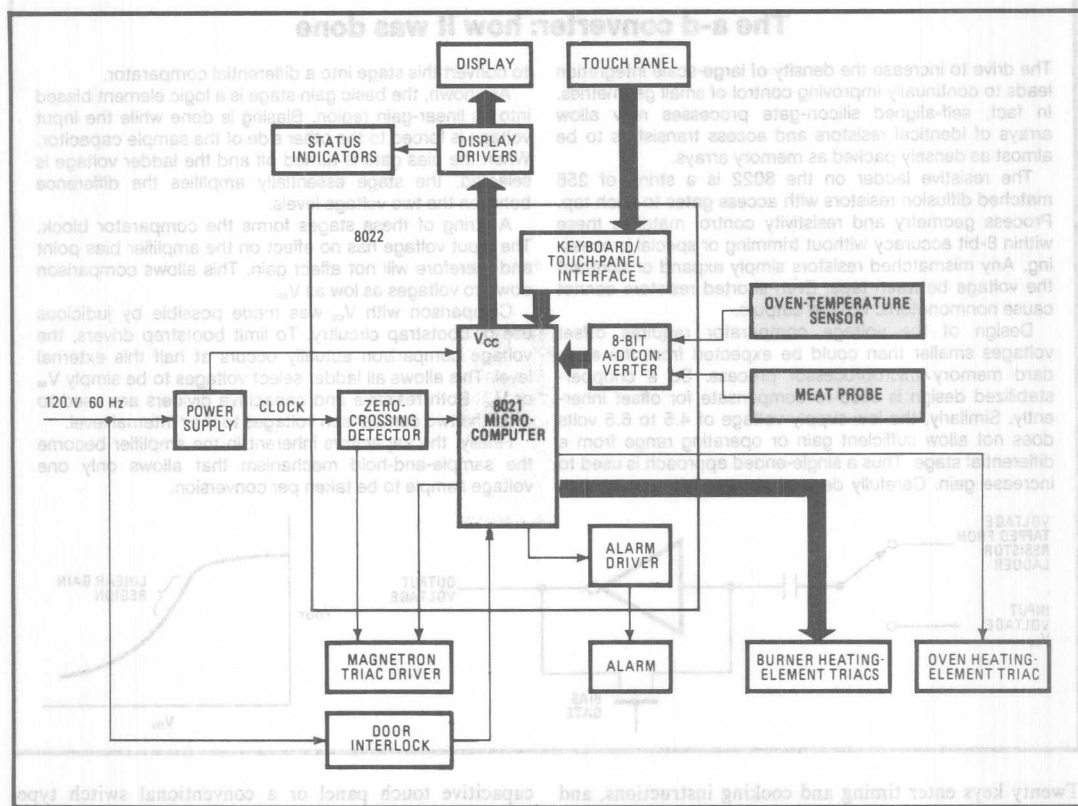
However, the switch point can be both tightly controlled and adjusted by specially biasing V_{th}. Uses for this would include high-noise-margin inputs (up to V_{cc}/2), unusual logic-level inputs as from a diode-isolated keyboard, analog-channel extension, and direct interfacing of capacitive touch panels. The comparator action is automatic, and the port is read just as is any other port.

Three advantages

Since the on-chip comparators allow small voltage changes to be detected, a cost-effective and safe touch panel can be built. Many appliances using touch panels have as much as 100 volts at the panel, albeit with extremely low power. The comparators in the 8022, however, permit appliance touch panels to be operated in the 10-to-15-v range.

The advantages of a low-voltage touch panel are three. First, it costs less to generate and switch the lower voltage. Then, since the keyboard operates at below 30 v, it is an Underwriters Laboratories' class II system, which can sharply cut the time required for approval. Finally, the possible product-liability problems associated with high-voltage operation disappear.

Simplified capacitive touch-panel operation is shown in Fig. 3. Contact with the panel drives both the voltage buffer and input to ground. When port 0 is read, a 0 on any line indicates a touched switch. The microcomputer drives the voltage buffer to recharge the panel. Matrix



5. Oven controller. The use of the 8022 is demonstrated in this controller for a combination microwave and conventional oven. The chip needs no assistance in figuring temperatures from thermistors connected to its analog inputs, reading inputs from a touch panel, detecting zero-crossing of ac for firing triacs and gating clocks and timers, direct-driving an alarm, and storing cooking-time instructions.

switch panels may also be sensed by the comparators.

Each pin on port 0 may or may not have an internal pull-up resistor: the option is chosen during selection of the ROM program code. If a resistor is left out for a given pin, the output appears as a true open drain for the range V_{ss} to V_{cc} . There is no temporary low-impedance drive to V_{cc} , as is the case with the remaining quasi-bidirectional ports. With open drains, accurate output waveforms can be generated, and operational amplifiers can be driven directly, for example.

The zero-crossing detector

Although the T_1 test pin on the 8022 may be driven directly by a digital input, it has special circuitry to detect an ac signal crossing its average direct-current level. The signal required for the zero-cross detection mode must be 2 to 4 v peak to peak and have a maximum frequency of 1 kilohertz. It couples to T_1 through an external capacitor.

Figure 4 shows the waveforms for zero-crossing detection. The internal digital state of T_1 is sensed as a 0, until the wave's rising edge crosses the average dc level, when it becomes a 1. The digital transition takes place within a 5° phase from the zero point. The digital level then remains at 1 until the input goes approximately 100 mv

below the zero point on the falling edge. The 100-mv hysteresis keeps noise from causing chattering of the internal signal.

The zero-crossing detection capability allows the applications designer to make the 60-hertz power signal the basis for system timing. All timing routines, including time of day, can be implemented with the signal and just a few conditional jump instructions.

Moreover, since T_1 is also an input to the external event counter, the detection feature may be combined with this counter to interrupt processing at the critical zero-crossing point. Thus the user can trigger phase-sensitive devices, such as triacs and silicon-controlled rectifiers, and use the 8022 in such applications as shaft-angle measurement and speed control of motors—anywhere that the zero crossing of a waveform provides timing information.

An oven controller

The 8022's high level of functional integration provides a single-chip solution to sophisticated, high-volume controller applications that have required relatively expensive multichip designs. An example is a controller (Fig. 5) for a stove with a combined microwave and conventional oven and range-top burners.

The a-d converter: how It was done

The drive to increase the density of large-scale integration leads to continually improving control of small geometries. In fact, self-aligned silicon-gate processes now allow arrays of identical resistors and access transistors to be almost as densely packed as memory arrays.

The resistive ladder on the 8022 is a string of 256 matched diffusion resistors with access gates to each tap. Process geometry and resistivity control matches these within 8-bit accuracy without trimming or special processing. Any mismatched resistors simply expand or contract the voltage between taps. Even shorted resistors cannot cause nonmonotonic voltage outputs.

Design of the voltage comparator requires offset voltages smaller than could be expected from the standard memory/microprocessor process. So a chopper-stabilized design is used to compensate for offset inherently. Similarly, the low supply voltage of 4.5 to 6.5 volts does not allow sufficient gain or operating range from a differential stage. Thus a single-ended approach is used to increase gain. Carefully devised circuit tricks are enough

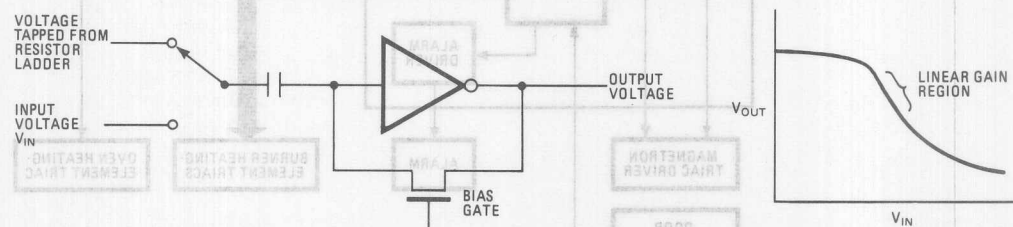
to convert this stage into a differential comparator.

As shown, the basic gain stage is a logic element biased into its linear-gain region. Biasing is done while the input voltage is forced to the other side of the sample capacitor. When the bias gate is turned off and the ladder voltage is selected, the stage essentially amplifies the difference between the two voltage levels.

A string of these stages forms the comparator block. The input voltage has no effect on the amplifier bias point and therefore will not affect gain. This allows comparison down to voltages as low as V_{ss} .

Comparison with V_{cc} was made possible by judicious use of bootstrap circuitry. To limit bootstrap drivers, the voltage comparison actually occurs at half this external level. This allows all ladder select voltages to be simply V_{ss} or V_{cc} . Both resistive and capacitive dividers are used to drop the two comparison voltages to their internal level.

Finally, the capacitors inherent in the amplifier become the sample-and-hold mechanism that allows only one voltage sample to be taken per conversion.



Twenty keys enter timing and cooking instructions, and a four-digit display shows cooking time, temperature, and the time of day. Two temperature-sensing thermistors are employed, one for standard use and the other for microwave use.

While such a system could be controlled by a conventional 4-bit or 8-bit microcomputer, external circuitry would be required to interface the keyboard, convert the analog signals to digital data, drive an audio alarm, and determine the zero-crossing point of the 60-Hz power wave for timing functions and magnetron control. The 8022 reduces this multichip system to a single chip. The computer-plus-converter chip can save the oven maker upwards of several dollars in parts costs.

In this application, the 8022 program memory stores all control programs, cooking and power-cycling algorithms, and timing routines. Its 2-kilobyte ROM is large enough to provide for easy expansion of oven features and product differentiation. The on-chip RAM stores temperatures, power-level and timing settings, and all intermediate computational results.

The analog signals from the conventional temperature sensor and the microwave meat probe feed directly into the two analog inputs on the 8022 without any additional circuitry. What's more, the chip's 8-bit a-d converter gives more accurate temperature sensing than most existing discrete configurations.

The keyboard interfaces directly to the device through port 0. The keyboard in this application can be either a

capacitive touch panel or a conventional switch type, since the 8022 directly interfaces either.

The T_1 pin in the zero-crossing detection mode establishes an accurate time base for all timing routines, including cooking cycles, presetting functions, and time of day. To accomplish this, the chip detects a zero crossing using the two conditional-jump instructions associated with T_1 : JT_1 and JNT_1 . Then it increments a register in data memory, effectively keeping track of elapsed time. Using this technique, a time-of-day routine can be written for most applications in less than 30 bytes of code.

Control of the magnetron

The zero-crossing detection capability also efficiently controls the microwave's magnetron. To minimize current surges through the system, the magnetron should be fired at the peak of the ac wave (90°). To achieve this performance, the 8022 detects the zero crossing point with its T_1 pin and delays the 90° phase shift with the internal timer.

The high-current drive pins, Pl_0 and Pl_1 , are tied together to directly drive a piezoelectric alarm, which requires 10 to 15 mA of current. The remaining I/O lines are used to drive the display and status indicators, to monitor the door interlock, and to control the triacs that switch the burner and oven heating elements. The internal timer controls the refreshing of the displays and the scanning of the keyboard. □

Microcomputer's on-chip functions ease users' programming chores

The one-chip 8022 includes hardware, such as an a-d converter, that combines with the instruction set for easy development of routines

by William F. Ittner and Jeffrey A. Miller, Intel Corp., Santa Clara, Calif.

□ A single-chip microcomputer that incorporates analog-to-digital conversion, comparator inputs, and ac zero-crossing detection is a strong candidate for low-cost, high-volume applications. Moreover, to maintain its front-runner position, the new 8022 has been designed for ease of programming: many common routines are invisible to the user because they are performed in on-chip hardware.

The 8022's instruction set, in conjunction with its hardware features, affords programming ease in the development of routines for translating analog signal levels, monitoring temperatures, reading capacitive-touch-panel inputs, controlling phase-sensitive thyristors, and calculating the time of day. For example, performing an a-d conversion requires software only to select the appropriate analog input; the actual conversion is performed entirely in hardware. This leaves room in the program memory for additional system functions. Furthermore, the instruction set accommodates bit handling, binary and binary-coded-decimal arithmetic, and direct table look-up, and it has extensive facilities for input selection and input-based program jumps.

The 8022 [*Electronics*, May 25, p. 122] is the first general-purpose single-chip microcomputer to offer an on-chip a-d converter. While retaining the 8-bit central processing unit, 64 bytes of random-access memory, clock, zero-crossing detection, and timer/event counter featured in its 8021 predecessor, the new chip doubles the read-only memory to 2 kilobytes and provides comparator inputs on eight input/output lines, five more digital I/O lines (including an extra test pin), full interrupt capability, and two 8-bit a-d input channels. Such a decrease in system component count cannot help but minimize cost and increase reliability.

Easy a-d conversion

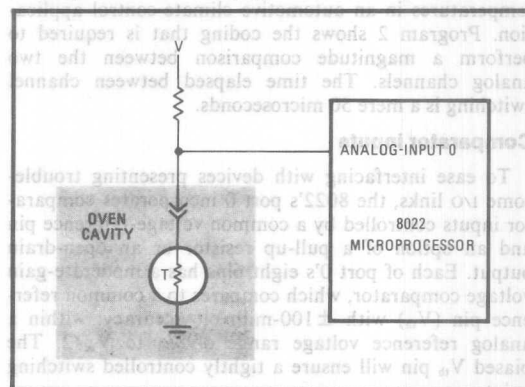
The 8022's a-d converter has two multiplexed channels, selectable with the SEL AN0 (select analog input 0) or SEL AN1 (select analog input 1) instructions. Built-in successive-approximation hardware accomplishes the conversion. The select instructions and the RAD command (read a-d conversion result) are the only software instructions necessary. The select instructions restart the continuously occurring conversion process, but do not affect the conversion-result register. The new valid digital value can be read from the CRR during the

fourth cycle after a select instruction and every fourth instruction cycle thereafter.

An application that points up the advantages of this easy-to-use on-chip converter is monitoring temperature in an oven controller. The temperature is sensed by a thermistor probe located in the oven (Fig. 1). In such a system, noisy analog signals are apt to prevail, obscuring the readings. But since so few instructions are needed for each sampling, a software filtering technique can be added at little expense for increased accuracy. One software filtering method is to average each reading with the previous samples:

```
SEL AN0      ;Start conversion
MOV R0, #30   ;Point to storage location of previous
              ;a-d sample average
RAD          ;Read second sample result
ADD A, @R0    ;Add last sample to new sample
RRC A        ;Divide by 2
MOV @R0, A    ;Store new average
```

Excessive noise may require averaging of many readings taken over a short period of time. Program 1 illustrates a method of computing the average of 16 readings. In such averaging, it is necessary to select the



1. Talk about simple. To sense temperature with the 8022, all that is needed is a thermistor pulled up to the supply. Simpler yet are the instructions to sense the voltage divider's potential; select analog input 0 (SEL AN0), and read conversion-result register (RAD).

	SEL	AN0	select & start conversion
	MOV	R2, #16	16 readings
LOOP:	RAD		read result
	ADD	A, @R0	LSB
	MOV	@R0, A	save new LSB
	CLR	A	
	ADDC	A, R4	MSB add carry to R4
	MOV	R4, A	save new MSB
	DJNZ	R2, LOOP	next reading
	SWAP	A	MSB into MSN
	XCH	A, @R0	
	SWAP	A	divide by 16
	XCHD	A, @R0	location 26 in RAM now contains the average value of 16 conversions over a period of 1.44 msec
	SEL	AN1	start other conversion
	CPL	A	
	INC	A	
	MOV	@R0, A	save first conversion
	RAD		read second conversion
	ADD	A, @R0	add first conversion A equals the differential in ones complement
	JZ	EQUAL	AN0 = AN1
	JC	LESTHN	AN0 < AN1
			AN0 > AN1

appropriate analog channel only once. Thereafter, a new conversion result is available every four instruction cycles.

Often noise on the analog input is due to 60-hertz pickup. To minimize this interference, analog signals should be synchronized with the line voltage, accomplished in the 8022 by the on-board zero-crossing-detection circuitry. The combination of line synchronization with simple filtration renders digital values impervious to line-generated noise.

Signal averaging may be used for more than noise filtering. Since it can be applied to either channel 1 or channel 0 (whichever is selected with a SEL ANx instruction), each channel may monitor different functions in one system, such as temperature and pressure in a process-control application. The fast a-d conversion time permits rapid switching between the two channels.

A similar software technique permits measuring the same variable in two different locations and comparing the two results, as in checking the internal and external temperatures in an automotive climate-control application. Program 2 shows the coding that is required to perform a magnitude comparison between the two analog channels. The time elapsed between channel switching is a mere 50 microseconds.

Comparator inputs

To ease interfacing with devices presenting troublesome I/O links, the 8022's port 0 incorporates comparator inputs controlled by a common voltage-reference pin and an option of a pull-up resistor or an open-drain output. Each of port 0's eight pins has a moderate-gain voltage comparator, which compares to a common reference pin (V_{th}) with ± 100 -millivolt accuracy, within a analog reference voltage range of V_{ss} to $V_{cc}/2$. The biased V_{th} pin will ensure a tightly controlled switching point.

A typical use for port 0 is in the interfacing with the capacitive touch panels on microwave ovens and other new appliances. A touch-panel switch consists of two capacitors in series. One lead is attached to a high-

voltage buffer (10 to 30 volts). The other is attached to the port 0 sense input. As a finger touches the common point, the drive signal is shunted by body capacitance, attenuating the signal reaching the input.

Low-voltage touch-panel operation (less than 30 v) is possible, since the comparators allow small voltage changes to be detected. Most of the present touch-panel designs require a 50-to-100-v drive on the touch panel.

Capacitive touch panels can be multiplexed in the same manner as can mechanical keyboards (Fig. 2). The vacuum fluorescent display and the touch panel are integrated to optimize hardware through shared high-voltage buffers.

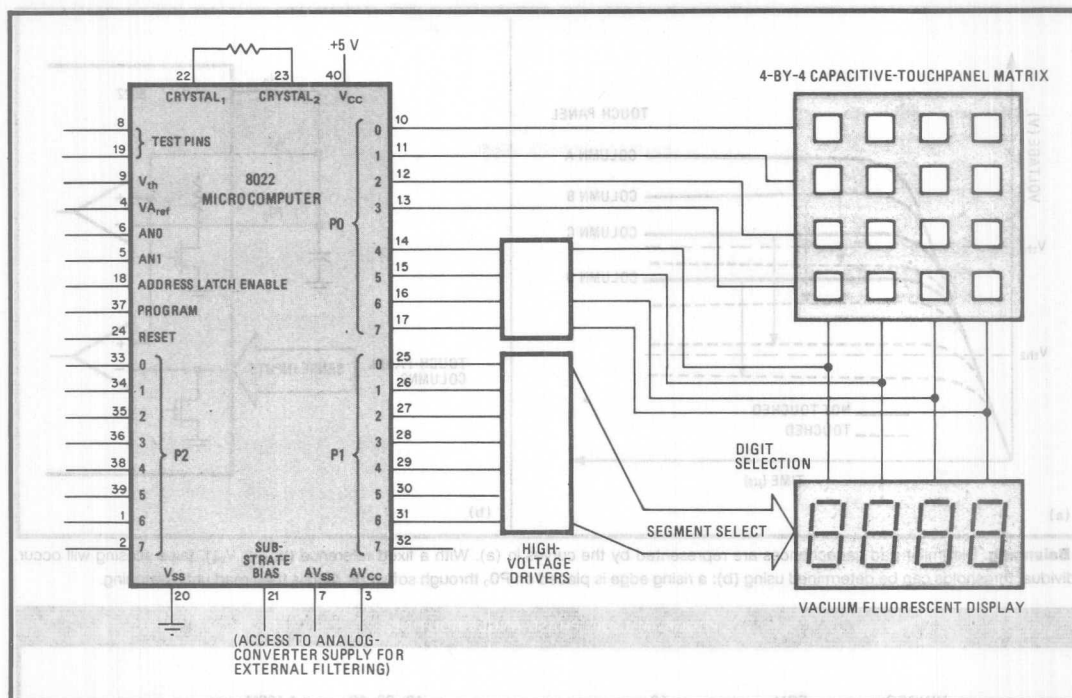
Program 3 lists the software that is necessary to refresh the display and scan the touch-panel matrix. This routine could be adapted to serve as part of a timer/interrupt scheme that would generate an interrupt at precise intervals for a flicker-free display. Another portion of the software would check for any touched input pads, test for valid entry, and enter key-depression codes into the main program.

Correcting pad imbalance

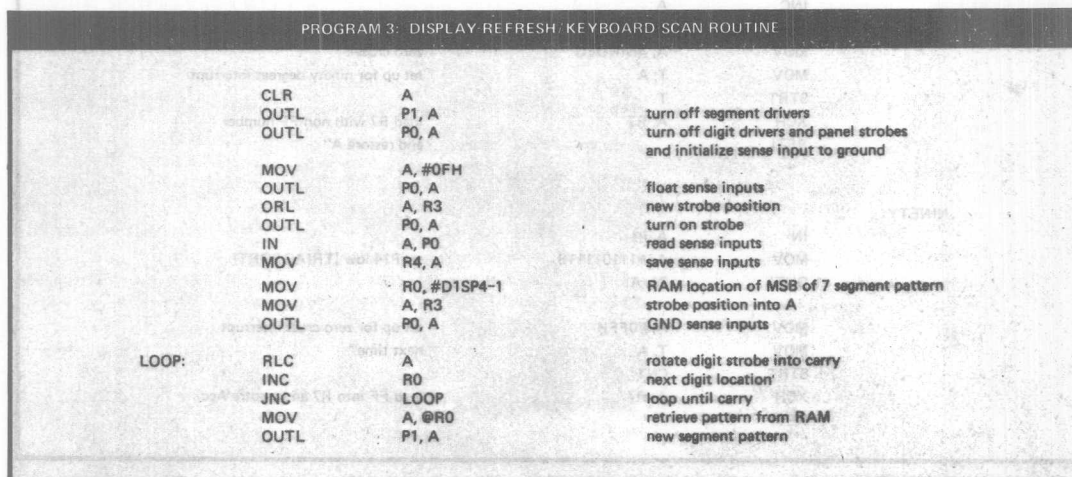
A common problem with capacitive touch panels is their imbalance. Layout process, aging, and surface impurities all cause the capacitance to vary from touch pad to touch pad, resulting in a family of curves (Fig. 3a) of voltage levels from each column of touch pads reaching the sense inputs. As the curves show, if threshold voltage V_{th1} alone were used, one column would always appear touched; if V_{th2} were used exclusively, three of the columns would never appear touched.

To compensate for such varying capacitance levels, the on-chip analog-input circuit may be used to allow multiple input voltage levels. Figure 3b depicts the 8022 version of such a circuit. AN0 and V_{th} are tied together with a capacitor to line 0 of port 0. The pull-up resistor option is used on line 0 to provide an RC timing network connected to AN0, V_{th} , and P0₀. The remaining seven lines of port 0 are the sense lines for the touch panel and use the open-drain-output option.

Handling the different voltage levels would begin with initializing the data by plotting the family of curves with the AN0 input. This is done by writing a 0 to P0₀ (grounding P0₀) which initializes V_{th} to 0 v. A logic 1 is then written to P0₀, which begins to pull the RC network



2. Multiplexed touch panel. A capacitive touch panel and high-voltage display may be combined in much the same way as a mechanical keyboard and light-emitting-diode array. To save hardware, an obvious choice is to share the high-voltage drivers.

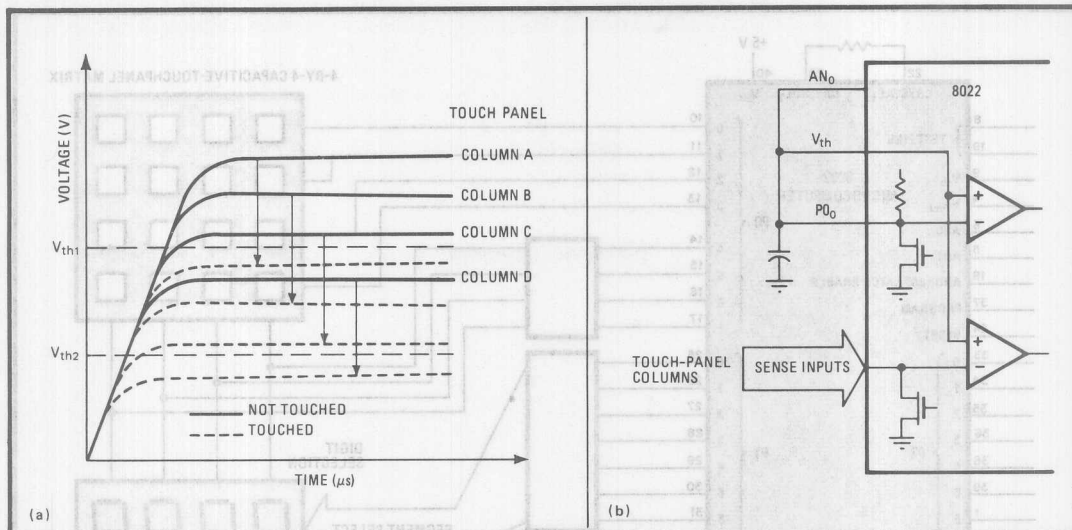


toward 5 v. As V_{th} ramps upward, the sense lines are monitored for input changes, which will go from 1 to 0 as the not-touched voltage curve for each input is intersected. As the changes occur, the a-d value for each sense line can be read and stored.

Thus the threshold reference voltage for each sense line can be determined by establishing the not-touched voltage levels and by placing the threshold reference voltage below this level. As each row of the keyboard is

scanned, the RC network is initialized to 0 v and ramps upward, varying the V_{th} level. The a-d converter monitors this level looking for the calculated threshold points. As the points are intersected, the corresponding sense inputs are read, with a 0 indicating a touched input and a 1 indicating a not-touched input. Thus, a multiplexed capacitive touch panel can be scanned and balanced without adding external components to the inputs.

For systems requiring more than two analog inputs to



3. Balancing. Dissimilar pad capacitances are represented by the curves in (a). With a fixed reference (V_{th1} or V_{th2}), false sensing will occur. Individual thresholds can be determined using (b): a rising edge is placed on $P0_0$ through software; AN_0 is then read until switching.

PROGRAM 4: 90° PHASE-ANGLE ROUTINE			
NINDEG	EQU	13	13x32x10 usec = 4.160M msec
LOC 7:	XCH	A, R7	save Acc and get flag byte
	INC	A	
	JNZ	NINETY	is it zero cross or 90 deg.
	MOV	A, #NINDEG	zero cross"
	MOV	T, A	set up for ninety degrees interrupt
	STRT	T	
	XCH	A, R7	load R7 with non-FF number
	RETI		and restore A"
NINETY:	IN	A, P1	
	MOV	A, #1101111B	set P14 low (TRIAC PORT)
	OUTL	P1, A	
	MOV	A, #0FFH	set up for zero cross interrupt
	MOV	T, A	next time"
	STRT	CNT	
	XCH	A, R7	load FF into R7 and restore Acc
	RETI		

the 8022, port 0's comparator may be reconfigured to permit forming of pseudo-analog inputs from variable-threshold digital inputs. The hardware configuration can be identical to that of Fig. 3b. In this scheme, sense inputs act as additional analog inputs of less accuracy than AN_0 and AN_1 (about 6 bits).

The sequence for implementing the extension is essentially the same found in the variable-threshold touch panel. As V_{th} ramps upward, a port 0 bit is monitored for a change from 1 to 0. At the change, AN_0 is read,

corresponding to the value of the analog input into port 0 (with some error due to the time lag, which can be subtracted). This configuration can be utilized when it is possible to trade off accuracy for cost improvements: one analog input with 8-bit accuracy and seven analog inputs with 6-bit accuracy. Such may be the case in a range controller that monitors temperature in two ovens, a meat probe, and two of the four burners, all to an accuracy within 10°F.

To establish a reliable time base and to switch ac

PROGRAM 5: COMPUTING THE TIME OF DAY

	ORG	7	T1 timer interrupt vector
	MOV	A, #0FFH	initialize timer
	MOV	T, A	
	MOV	R0, #TIME0 - 1	TIME0 = LSB of timer register
	MOV	R1, #TABLE	LSB of ROM look-up
LOOP:	INC	R0	point to next byte
	MOV	A, @R0	retrieve BCD byte
	ADD	A, #1	increment
	DA	A	decimal adjust
	MOV	@R0, A	restore BCD byte
	MOV	A, R1	test for carry
	MOVP	A, @A	using table entry
	XRL	A, @R0	and"
	INC	R1	bump pointer
	JNZ	DONE	no carry, wait for next tick
	XCH	A, @R0	carry, set digit pair to 00
	ANL	A, #1	was overflow hours?
	JZ	LOOP	no, increment next byte
	MOV	@R0, A	yes, set hours to 1
DONE:	RET		
TABLE:	DB	60 H	sixtieth
	DB	60 H	minutes
	DB	13 H	hours use 25 for 24 hour operation

loads, the 8022 has circuitry built into the T_1 pin to detect an ac signal crossing its average dc level. The switching is at predetermined points of the sine wave to reduce inrush currents or radio-frequency interference.

Zero-crossing detection

There are several methods by which software can monitor the input. The simplest method involves the jump instructions JT_1 and JNT_1 , which correspond to jump on T_1 high, and jump on T_1 low, respectively. The rising edge of the T_1 input is the most accurate: the falling edge contains 100 mv of hysteresis to increase noise margin. The two jump instructions can be used back to back to find this zero-crossing point:

```
HERE1: JT1 HERE1 ;Wait here if line high
HERE2: JNT1 HERE2 ;Wait here if line low
        ; Zero cross
```

To reduce loop time, the T_1 pin may also be coupled to the event counter. The start-counting instruction couples the rising edge into the 8022's internal 8-bit timer. With each rising edge, the timer increments by one, and when it increments from FF Hex to 00, an overflow flag is set. If the interrupt line is activated, an interrupt vector at location 7 will occur. Since the timer may be preloaded with any value, it is possible to cause an interrupt to occur on the next zero crossing rather than waiting in the jump loop.

The following routine will initialize the timer to accomplish this. All other processing may be performed

while waiting for the zero crossing. The timer could be reloaded with FF Hex during the interrupt routine to generate an interrupt on each zero crossing.

```
MOV A, #0FFH ;Full count into accumulator
MOV T, A ;Load timer
STRC CNT ;T1 pin is source to timer
EN TCNTI ;Enable timer interrupt
```

Of course, the zero crossing is not always the best point to gate a control device. For example, an application involving a highly inductive device such as a magnetron transformer will produce inrush currents that are at their maximum at the zero-crossing point.

Low inrush

To minimize inrush in such a system, the triac is turned on at a 90° phase angle in the 60-Hz sine wave. Program 4 provides the software necessary to accomplish this task. The timer detects the zero-crossing point and times out to the 90° point, where the leading current will just be at a minimum. An interrupt occurs at both the zero and 90° points to prevent interference with normal processing. Both interrupts use the same interrupt vector location. Software determines the source of the interrupt and acts accordingly.

Another use of the T_1 input is generating the timing base for a time-of-day routine. The software implementing this routine is in program 5. The time parameters listed in the accompanying data table could be modified to accommodate either 12- or 24-hour operation. □

Designing with Intel's 8022 Microcomputer

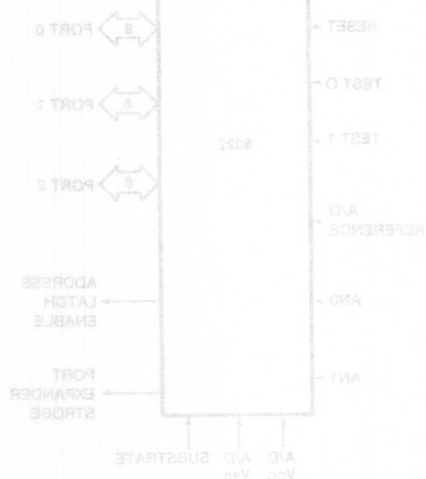


Figure 2. Logic Symbol

Product Overview

The heart of the 8022 is the Intel 8021, a general purpose single chip microcomputer, which is a lower performance, lower cost version of the 8048. Added to this central core are interrupts, additional I/O, and linear functions. Like the 8021, the 8022 is designed to operate over a power supply range of 4.5 to 5.5 volts.

The 8022 instruction set contains over 70 instructions and is a subset of the 8048 instruction set. To conserve memory and maximize throughput, most instructions are single-byte, single-cycle. No instructions are longer than two-byte, two-cycle. The instruction cycle time is 10 microseconds at a 3MHz clock rate. Extensive conditional branch logic is built into the processor to increase the overall efficiency of the instruction set for control applications. As examples, the JNZ instruction (branch if not zero) allows loops to be formed in just one instruction and the MOV A, @A allows single instruction table look-up of constants from program storage. Program storage in the 8022 consists of 2048 eight-bit bytes of mask programmable ROM.

Hardware stack and data memory are integrated in the 8022 RAM to enhance processing flexibility and memory utilization. The first eight RAM locations are designated as working registers and are directly addressable by any of the 11 direct register instructions. Besides increasing the variety of operations that can be performed on data in memory, this approach further reduces the number of instruction bytes required for processing. In addition to being used as working registers, Registers 0 and 1 can be used as Pointer registers to indirectly address all locations in memory using the indirect register instructions.

Contents

INTRODUCTION	3-230
PRODUCT OVERVIEW	3-230
PRODUCT FEATURES	3-231
System Clock	3-231
Timer/Counter	3-234
Test and Interrupt Inputs	3-235
Analog to Digital Converter	3-237
Port 0 Comparator Inputs	3-240
APPLICATION IDEAS	3-242
Power Supply Controller	3-242
DC Motor Control	3-244
Automotive Dashboard	3-245
Darkroom Timer	3-246
CONCLUSION	3-246

Figure 1. Pin Configuration

INTRODUCTION

Taking advantage of the latest advances in silicon technology, Intel has developed a complete control system on a chip, the Intel 8-bit microcomputer with an A/D converter on-chip. Whereas in the past microcomputers relied on external circuits for analog interfacing, it is now possible to build a one chip control system with analog interfacing, digital interfacing, and computer processing capabilities. Tackling the high volume, low cost controller market, the Intel 8022 microcomputer fits cost and space sensitive applications such as automobiles, appliances, and consumer products previously dominated by electromechanical controls. Its use, however, is not confined only to these applications. In medium volume applications, the 8022 provides the system designer with a simplified solution to many control problems. No longer is it necessary to expand valuable engineering time designing wheel spokes and axles; the whole car is available.

This note is intended to answer some design questions concerning the 8022 and to suggest to the reader possible applications and system configurations. The reader should refer to the 8022 Data Sheet for electrical specifications and details. It is also suggested that the reader consult with the MCS-48 User's Manual (July 1978 or later) for a complete description of the entire MCS-48 family of microprocessors of which the 8022 is the newest member.

The note is divided into two main sections. The first is a product description of the 8022, including a detailed discussion of the main features, their characteristics and how to use them. The second section discusses several possible applications, their configurations and design considerations.

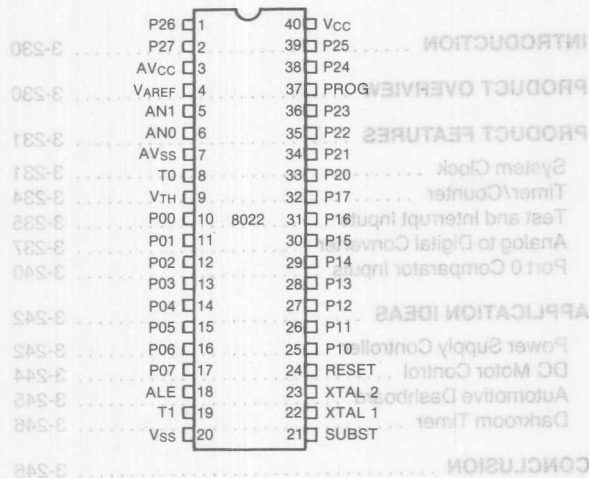


Figure 1. Pin Configuration

INTRODUCTION

Taking advantage of the latest advances in silicon technology, Intel has developed a complete control system on a chip, the 8022, the first 8-bit microcomputer with an A/D converter on-chip. Whereas in the past microcomputers relied on external circuits for analog interfacing, it is now possible to build a one chip control system with analog interfacing, digital interfacing, and computer processing capabilities. Tackling the high volume, low cost controller market, the Intel 8022 microcomputer fits cost and space sensitive applications such as automobiles, appliances, and consumer products previously dominated by electromechanical controls. Its use, however, is not confined only to these applications. In medium volume applications, the 8022 provides the system designer with a simplified solution to many control problems. No longer is it necessary to expend valuable engineering time designing wheel spokes and axles; the whole cart is available.

This note is intended to answer some design questions concerning the 8022 and to suggest to the reader possible applications and system configurations. The reader should refer to the 8022 Data Sheet for electrical specifications and details. It is also suggested that the reader consult with the MCS-48 User's Manual (July 1978 or later) for a complete description of the entire MCS-48 family of microprocessors of which the 8022 is the newest member.

The note is divided into two main sections. The first is a product description of the 8022, including a detailed discussion of the main features, their characteristics and how to use them. The second section discusses several possible applications, their configurations and design considerations.

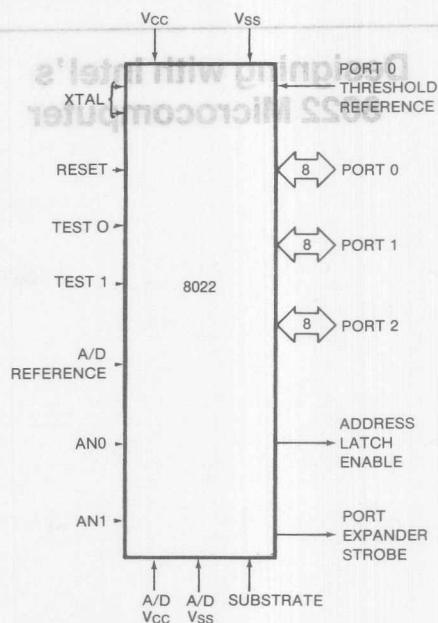


Figure 2. Logic Symbol

Product Overview

The heart of the 8022 is the Intel 8021, a general purpose single chip microcomputer, which is a lower performance, lower cost version of the 8048. Added to this central core are interrupts, additional I/O, and linear functions. Like the 8021, the 8022 is designed to operate over a power supply range of 4.5 to 6.5 volts.

The 8022 instruction set contains over 70 instructions and is a subset of the 8048 instruction set. To conserve memory and maximize throughput, most instructions are single-byte, single-cycle. No instructions are longer than two-byte, two-cycle. The instruction cycle time is 10 microseconds at a 3MHz clock rate. Extensive conditional branch logic is built into the processor to increase the overall efficiency of the instruction set for control applications. As examples, the DJNZ instruction (decrement register and jump if not zero) allows loops to be formed in just one instruction and the MOVP A, @A allows single instruction table look-up of constants from program storage. Program storage in the 8022 consists of 2048 eight bit bytes of mask programmable ROM.

Hardware stack and data memory are integrated in the 64 byte RAM to enhance processing flexibility and memory utilization. The first eight RAM locations are designated as working registers and are directly addressable by any of the 11 direct register instructions. Besides increasing the variety of operations that can be performed on data in memory, this approach further reduces the number of instruction bytes required for processing. In addition to being used as working registers, Registers 0 and 1 can be used as Pointer registers to indirectly address all locations in memory using the indirect register instructions.

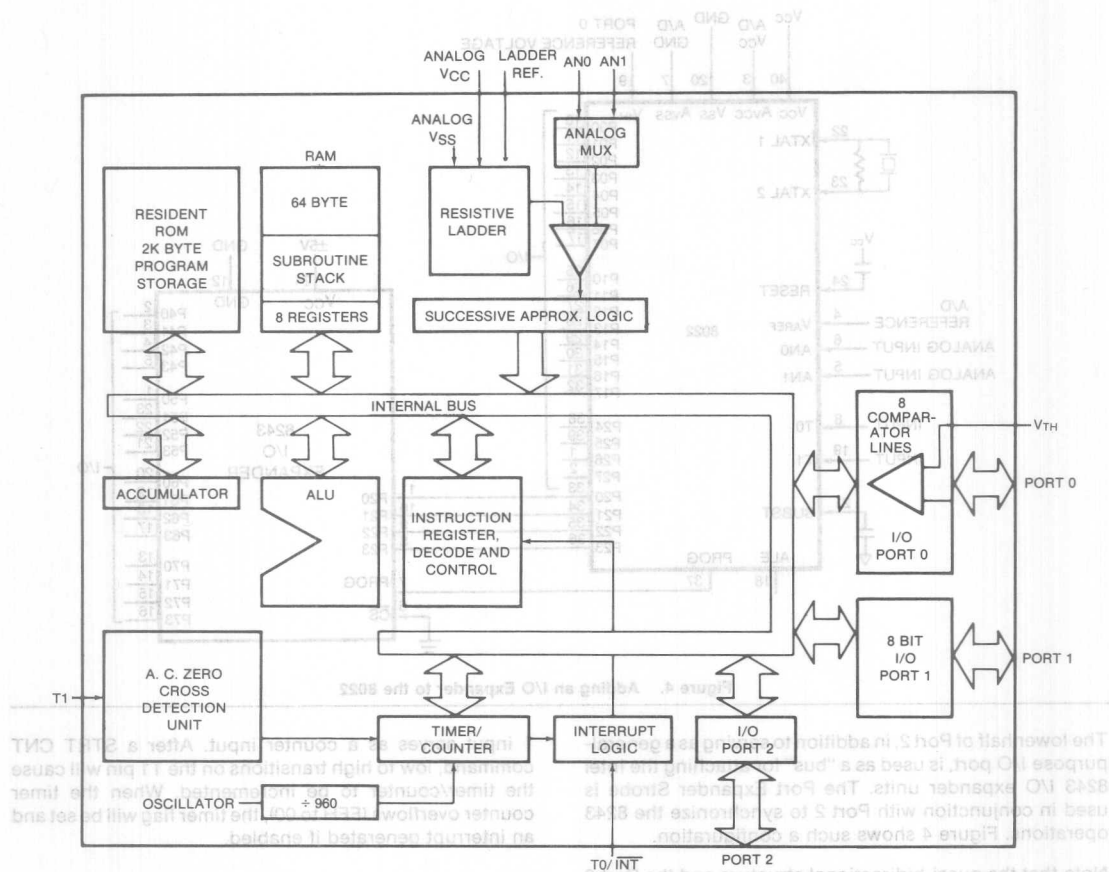


Figure 3. 8022 Block Diagram

The next 16 bytes of RAM may be used as the address stack to enable the processor to keep track of the return addresses generated from instructions and in handling interrupts. Since two bytes are needed to store each address, the 16 bytes of address stack allow up to a total of eight levels of subroutine nesting. A 3-bit stack pointer supplies the address of the locations to be loaded with the next return address generated. This stack pointer is incremented when a return address is stored and decremented when an address is fetched during a subroutine or interrupt return. If all eight levels of subroutine nesting are not required by an application, the unused portion of the address stack may be used as standard RAM.

The 8022 has an extremely flexible and powerful I/O structure. The 26 digital I/O lines are configured into three 8-bit general-purpose ports and two test pins, T0 and T1. All three ports are quasi-bidirectional, meaning all lines are useable as inputs or outputs on a line-by-line basis under software control.

To increase the user's flexibility, any line of Port0 can also be designated an open drain output by removing the pullup device present on the line via mask option. This is useful in driving analog circuits and interfacing to high impedance digital I/O. In addition to the open drain option, Port 0 has voltage comparator inputs with a common reference pin (V_{TH}). In appliance control and other applications, this allows direct glass touchpanel interfacing with relatively low voltage (10-15V) drive, thus limiting product liability problems and easing U.L. approval. The Port 0 comparator inputs are also generally useful in many other ways from expanding analog inputs to maximizing margin on noisy signals.

To further increase user flexibility and reduce system cost, two I/O pins (P10 and P11) have been designated as high current drive pins with the ability to sink 7ma each, instead of the standard TTL load of 1.6ma. This can eliminate the need for discrete drive transistors in many applications.

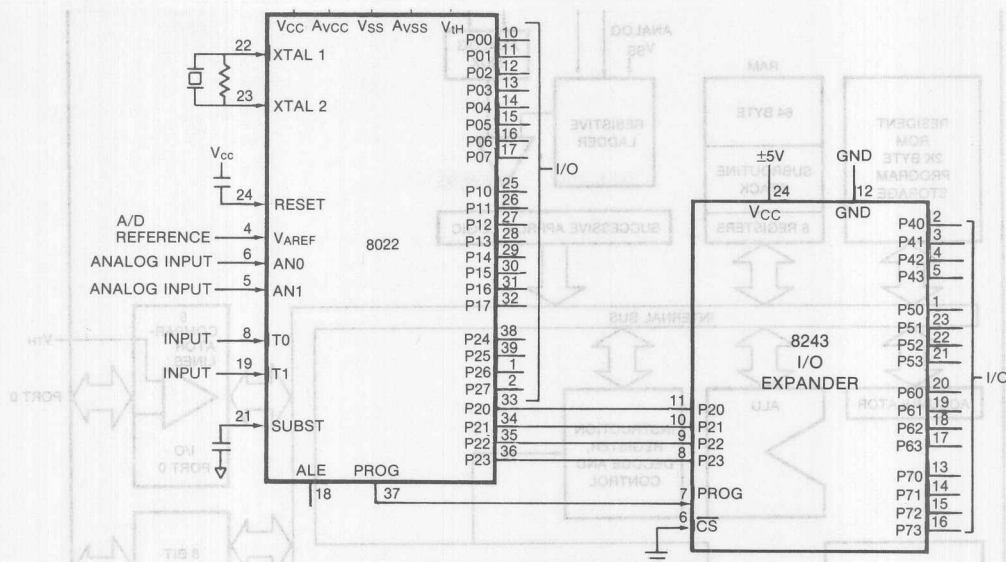


Figure 4. Adding an I/O Expander to the 8022

The lower half of Port 2, in addition to serving as a general-purpose I/O port, is used as a "bus" for attaching the Intel 8243 I/O expander units. The Port Expander Strobe is used in conjunction with Port 2 to synchronize the 8243 operations. Figure 4 shows such a configuration.

Note that the quasi-bidirectional structure and the Port 2 expansion bus are consistent with all MCS-48 products and are fully described in the MCS-48 User's Manual.

Frequently in control applications, the state of one or two signals must be monitored so that a fast response can be accomplished. The 8022's two test pins offer this capability. Both test pins, T0 and T1, are directly testable via two conditional branch instructions. The T0 pin can also cause an interrupt. The T1 pin, in addition to being directly testable, has the ability to detect the zero crossing of slowly moving AC inputs. This is useful in controlling 50/60Hz power. It also enables the 8022 to precisely control phase sensitive devices, such as triacs and SCRs. Again external circuitry is reduced.

The 8022 contains its own clock and oscillator circuitry and requires only an external timing control element to generate all internal timing signals. An inductor, a crystal, or an external clock may be used as the timing control device.

The programmable 8-bit timer/event counter enables the user to accurately monitor elapsed time by providing a hardware replacement for software overhead such as timing loops. Total count capacity is 8192 instruction cycles or 81.9 msec at a 10 microsecond cycle time. The timer may also be used as an event counter where the Test

1 input serves as a counter input. After a STRT CNT command, low to high transitions on the T1 pin will cause the timer/counter to be incremented. When the timer counter overflows (FFH to 00), the timer flag will be set and an interrupt generated if enabled.

The analog to digital converter is designed to simplify and cost reduce interfacing to analog sources. All parts of the converter are integrated onto the chip, with the exception of the voltage reference. Conversion is completely hardware controlled using a successive approximation technique and occurs in four instruction cycles or 40 microseconds. Three single byte instructions, SEL AN0 (select analog input 0), SEL AN1 (select analog input 1), and RAD (read A/D conversion result) are added to the 8021 instruction set to allow the programmer to interface to the converter conveniently.

Product Features

This next section will delve deeper into some of the functions which comprise the 8022 architecture. Chip architecture will be discussed along with design considerations, software routines, and hardware configurations. The specific items covered are CPU timing, the Timer/Counter, the TEST and Interrupt inputs, Zero Cross detection, the A/D converter, and the Port 0 comparator inputs.

System Clock

One of the first considerations in the system design is what frequency source should be used. The on-board oscillator can use a variety of elements to determine system fre-

quency. Depending on the accuracy needed, the element can be an inductor and capacitor, or a crystal and resistor. If necessary, the oscillator inputs can also be driven by an external source.

It should be noted that the values given in this section are approximate values based on a sampling of parts. In no case are these to be interpreted as guaranteed specifications. They are here as an aid in system design. Consult the final Data sheet or contact Intel direct if more information is needed for a critical design.

Inductor Mode

Figure 5 shows the proper configuration for the inductor mode. A parallel capacitor of 20 to 50pf is recommended for best frequency tolerance.

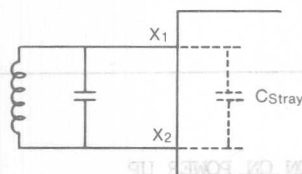


Figure 5.

Table 1 shows the effects of changes in parameters based on a sampling of parts. Part to part input capacitance differences (Cstray) will effect the tolerance. A less than 0.2% part to part tolerance can be expected with a parallel capacitance of 50pf. (see fig. #5). An additional 0.5% variation comes about when only 20pf is used in the tank circuit. This is because the stray capacitance in the 8022 and the PCB becomes a larger proportion of the total capacitance.

Vcc =	4.5v	5.5v	6.5v
f =	±0.2%	0	≈0.2%
Temp =	-40°C	25°C	85°C
f =	±0.6%	0	≈0.6%

Table 1. Inductor Mode

To determine the inductance and capacitance required for a given frequency, the equation

$$f = \frac{1}{2\pi\sqrt{LC}}$$

can be used. Due to the effects of stray capacitance the calculated frequency may be slightly high. It should be noted that the tolerances given in Table 1 do not include the tolerances of the inductor and capacitor used in the system. Mathematical analysis of the above equation will show that the frequency will change roughly proportional to the tolerances of L and C on a worst case situation. That is if both L and C are ±5% parts, the frequency will vary approximately ±5%.

Crystal Mode

Figure 6 shows the proper installation of a crystal. A one meg-ohm parallel resistor is required for operation with an 8021 or 8022. Application note AP-35 "CRYSTALS: Specifications for Intel Components" should be consulted for information on using and specifying crystals.

A 20pf capacitor is optional, but recommended, on X2. It has been found that using the capacitor increases the immunity of the microcomputer to line transient noise or spurious signals which may find their way into the system.

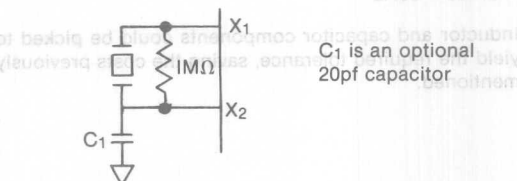


Figure 6.

Which One?

Which timing source to use is dependent on several factors. In most applications cost is of primary importance. The lowest cost device, but one which still gets the job accomplished, is the logical choice. Selecting the device which gets the job accomplished is the next task.

A Case Study

To exemplify the design tradeoffs in choosing a timing element consider the detection of 50Hz or 60Hz line frequency as may be needed in many consumer products being sold in the U.S. and overseas. Traditionally two products are produced, one for the U.S. market and one for the overseas market. A jumper selection to tell the processor which frequency source is being used is the only difference. This costs one I/O pin plus the costs of insertion and inventorying two products. All of these costs can be saved by allowing the processor to compute which frequency is coming in on the T1 pin. Figure 7 lists the software which could be used during a power-up routine to determine whether 50Hz or 60Hz timing should be used.

The timer is used to time the interval of one line cycle. If everything were perfectly accurate, one count would equal 50Hz while another count would equal 60Hz, but it's not. The power company frequency may shift slightly, plus the 8022 oscillator may drift as discussed earlier. The maximum allowable oscillator change must be calculated from the input source. Assuming the power companies may drift ±2 cycles, then the processor must be able to detect a difference of 58Hz-52Hz = 6Hz or less than 10.3% change. This means that the oscillator frequency itself cannot change more than 10.3% or ±5.15%. The crystal would definitely work but may be overkill. The Inductor/capacitor combination could be the most economical solution.

The equation

$$\frac{1}{\text{LF}} \times 30 \times 32 = \text{count}$$

where LF = line freq,
f = osc. freq.

will give the value of the time at the end of one line cycle. Plugging in the values for an oscillator of 3MHz $\pm 5\%$ and a ± 2 cycle deviation in line frequency, the counter will yield counts of:

$$47-56 = 60\text{Hz}$$

$$57-68 = 50\text{Hz}$$

Inductor and capacitor components could be picked to yield the required tolerance, saving the costs previously mentioned.

Timer/Counter

An 8-bit interval timer/counter is available to enable the user to keep track of time elapsed or number of events occurred while normal program execution and flow continues. The Auto 50/60Hz detection routine previously discussed is one of many possible applications of the timer/counter.

The timer/counter consists of a divide by 32 prescaler (only used in the timer mode) and an eight bit main timer. The STRT T command clears the prescaler and thereafter it increments once each 30 input clocks (once each single cycle instruction, twice each double cycle instruction). At the (1111) to (0000) transition the timer is incremented. A timer overflow from (FFH) to (00H) will set the timer flag along with the timer interrupt, if enabled (see below). A conditional branch instruction (JTF) is available for testing

LOC	OBJ	LINE	SOURCE STATEMENT
		1	;
		2	;
		3	;
		4	=====
		5	AUTO 50/60HZ DETECTION ON POWER UP
		6	=====
		7	PWRUP: CLR A ;CLEAR ACCM
		8	JMP PWRDET ;JUMP AROUND INTERRUPT ROUTINES
		9	;
		10	;
		11	INTERRUPT ROUTINES HERE
		12	;
		13	;
0014		14	ORG 20
		15	;
		16	MEASURE ONE LINE CYCLE
		17	=====
		18	PWRDET: JTI PWRDET ;WAIT FOR NEXT RISING EDGE
0014	5614	19	LINLOW: JNT1 LINLOW ;WAIT FOR NEXT RISING EDGE
0016	4616	20	MOV T,A ;CLEAR TIMER
0018	62	21	STRT T ;START TIMER
0019	55	22	;
		23	;
001A	561A	24	LINEHI: JTI LINEHI ;WAIT HERE FOR LINE TO GO LOW
		25	;
001C	461C	26	RISEDG: JNT1 RISEDG ;WAIT HERE FOR RISING EDGE
001E	65	27	STOP TCNT ;STOP TIMER AT END OF ONE LINE CYCLE
001F	42	28	MOV A,T ;READ TIMER VALUE
0020	03D1	29	ADD A,#-47 ;SUBTRACT 47
0022	E62C	30	JNC ORANGE ;ERROR-NOT WITHIN RANGE
0024	03F6	31	ADD A,#-10 ;SUBTRACT 10
0026	E62C	32	JNC HZ60 ;JUMP TO 60HZ ROUTINE
0028	03F4	33	ADD A,#-12 ;SUBTRACT 12
002A	F62C	34	JNC ORANGE ;ERROR-NOT WITHIN RANGE
		35	;
		36	HZ50: ;SET 50HZ FLAG
		37	;
		38	HZ60: ;SET 60HZ FLAG
		39	;
		40	ORANGE: ;LINE FREQUENCY ABNORMAL TRY AGAIN
		41	;
		42	;
		43	;
		44	;
		45	;

Figure 7.

All mnemonics copyrighted © Intel Corporation 1976.

this flag, the flag being reset each test. This instruction must also be used to initialize the timer overflow flag after a RESET; as RESET does not perform this function. Total count capacity for the timer is $2^5 \times 2^8 = 8192$ or 81.9 ms at a 10 micro second cycle time. Contents of the timer are moved to the accumulator by the MOV A,T instruction without disturbing the counting process. Conversely, the MOV T,A instruction loads the timer with the contents of the accumulator. Notice that the 8-bit timer can be read from and written to. The prescaler, however, can not. It is a separate 5-bit counter which is cleared only by a STRT T command.

The timer may also be used as an event counter. After a STRT CNT command the 8022 will respond to low-to-high transitions on the Test 1 pin by incrementing the timer. Transitions can occur no faster than once each three instruction cycles (every 30 microseconds when using a 3 MHz clock)—there is no minimum frequency. In this mode the prescaler is not used. The timer will contain the number of positive transitions occurring on T1 since a STRT CNT command.

The timer and event counter functions are mutually exclusive. Counting or timing may be started (STRT CNT, STRT T) or stopped (STOP TCNT) under program control.

The T1 pin, besides being an input to the counter, can also function as a testable input, detect the zero crossing of an AC signal, and interrupt processing. These functions, as well as those of the Test 0 pin and the interrupt structure, will be discussed in the next section.

Test And Interrupt Inputs

In addition to the 24 general purpose I/O lines which comprise ports 0, 1, and 2, the 8022 has two special inputs, T0 and T1, which are testable via conditional jump instructions. These pins allow inputs to cause program branches without the necessity to load an input port into the accumulator. The instructions JT0, JNT0, JT1, JNT1 will cause program flow to be modified depending on the state of the T0 or T1 pin. For instance, JT0 will cause a jump to the specified address if the T0 pin is high (a 1 level). Conversely, JNT0 will jump if T0 is low (a 0 level). If the jump does not occur, program flow continues with the next instruction.

The Test 0 pin serves as an external interrupt input as well as a testable input. An interrupt sequence is initiated by applying a low "0" level input to the T0 pin when the external interrupt is enabled (EN I). The interrupt is level triggered and active low to allow "WIRE ORING" of several interrupt sources at the input pin. When an interrupt is detected it causes a "call to subroutine" to location 3 in program memory as soon as all other cycles of the current instruction are complete. At this time, the program counter contents are saved in the program counter stack, but the remaining status of the processor is not.

Unlike the 8048, the 8022 does not contain a program status word. Thus, when appropriate, the carry and auxili-

ary carry flags must be saved by the software, as must be the accumulator. The routine shown below saves the accumulator and the carry flags.

Instructions	Comments
MOV R6,A	;save accumulator in register 6
CLR A	;clear accumulator
DA A	;convert carry flags into sixes
MOV R7,A	;save representation of carry flags

The end of an interrupt service subroutine is marked by the execution of a Return from Interrupt instruction (RETI). Prior to returning from the interrupt subroutine however, the status of the accumulator and the carry flags must be restored. The following routine restores the status of the accumulator and the carry flags, which were previously saved by the above program segment.

Instructions	Comments
MOV A,R7	;restore carry flags status to
ADD A,#0AAH	;accumulator and set/clear
	;carry flags
MOV A,R6	;restore accumulator
RETI	;return from interrupt

An interrupt or CALL to a subroutine causes the contents of the program counter to be stored in one of the eight register pairs of the Program Counter Stack. During a CALL instruction the program counter, when saved, points to the second byte of the CALL instruction (or the return address minus one). The stack contents are then incremented before being loaded into the program counter during a return (RET) from subroutine. During an interrupt the program counter, when saved, points directly to the return address. Thus, during a return (RETI) from interrupt, the stack contents are not incremented but loaded directly into the program counter. This difference makes it imperative to use only RETI's to return from interrupts, and RET's to return from subroutines.

The interrupt system is single level in that once an interrupt is detected all further interrupt requests are ignored until execution of a RETI re-enables the interrupt input logic. This sequence holds true also for an internal interrupt generated by timer overflow. If an external interrupt and an internal timer/counter generated interrupt are detected at the same time, the external source will be recognized first, if enabled. The timer/counter interrupt will be recognized, if enabled, after the return (RETI) from the external interrupt. Timer/counter generated internal interrupts and T0 generated external interrupts have separate vector locations. The external interrupt will vector to location 3, whereas an internal interrupt will vector to location 7.

If needed, a second external interrupt can be created by enabling the timer/counter interrupt (EN TCNTI), loading FFH into the counter (one less than terminal count) and enabling the event counter mode (STRT CNT). A low-to-high transition on the T1 input will then cause an interrupt vector to location 7.

can be used to detect the zero crossing point of slow moving AC signals. Execution of the STRT CNT instruction puts the T1 pin in the counter input mode by connecting T1 to the counter and enabling the counter. Subsequent low-to-high transitions on T1 will cause the counter to increment. Note that this operation differs from the rest of the MCS-48 devices, which increment the counter on high-to-low transitions. This change was made on the 8022 to take advantage of the accuracy of the rising edge detection on the zero cross circuitry.

When driven directly, this pin responds as a normal digital input. To utilize the zero cross detection mode, an AC signal of approximately 1-3 VAC p-p magnitude and a maximum frequency of 1kHz is coupled through an exter-

which is included in the T1 input. This circuit biases the T1 input exactly at its switching point, such that a small change will cause a digital transition to occur. This digital transition takes place within 5 degrees of the zero point.

The digital value of T1 remains a one until the falling edge of the AC input drops approximately 100mV below the switching point of the rising edge (100mV below the zero point, if the digital transition occurred exactly at the zero point). The 100 mV offset is created by hysteresis and eliminates chattering of the internal signal caused by external noise.

The accuracy of the zero crossing will be a function of the capacitor used (see Fig. 10). On critical systems the capacitor can be adjusted to improve overall accuracy.

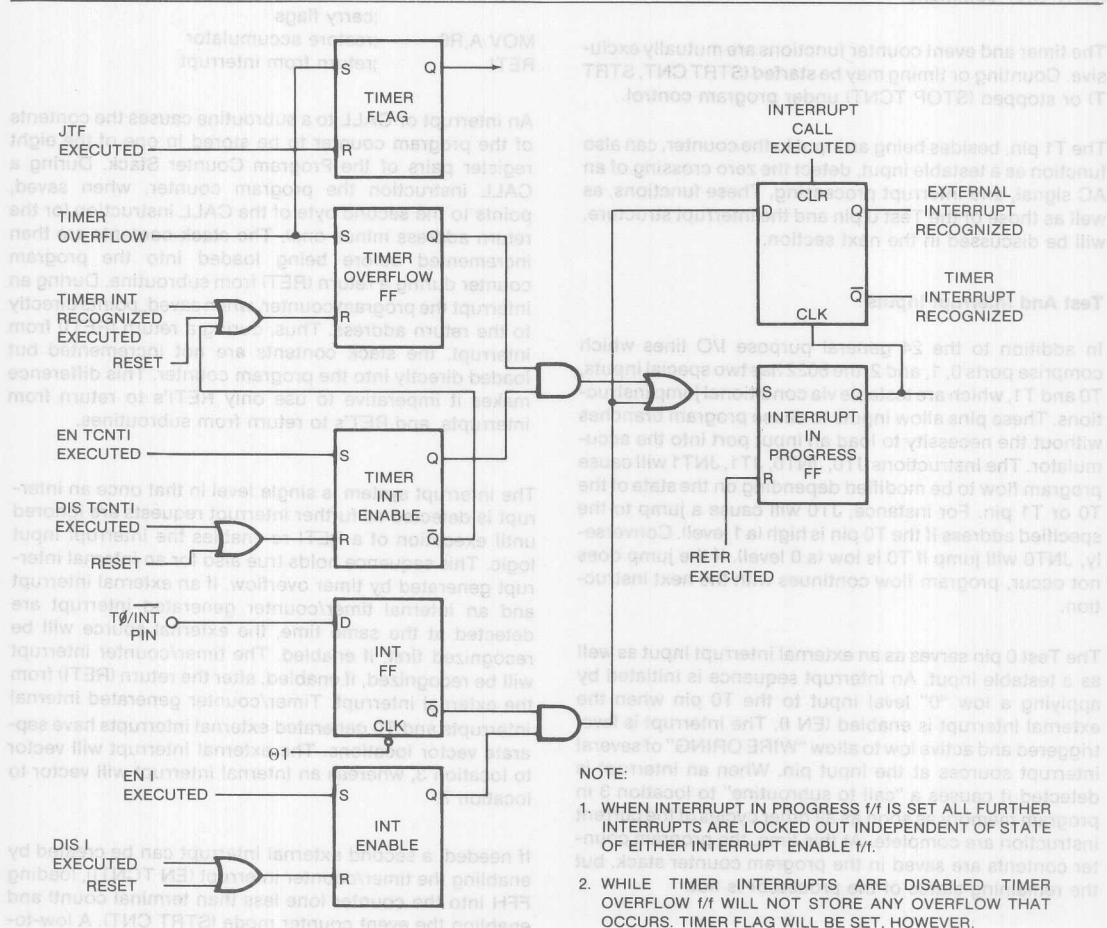


Figure 8. Interrupt Logic

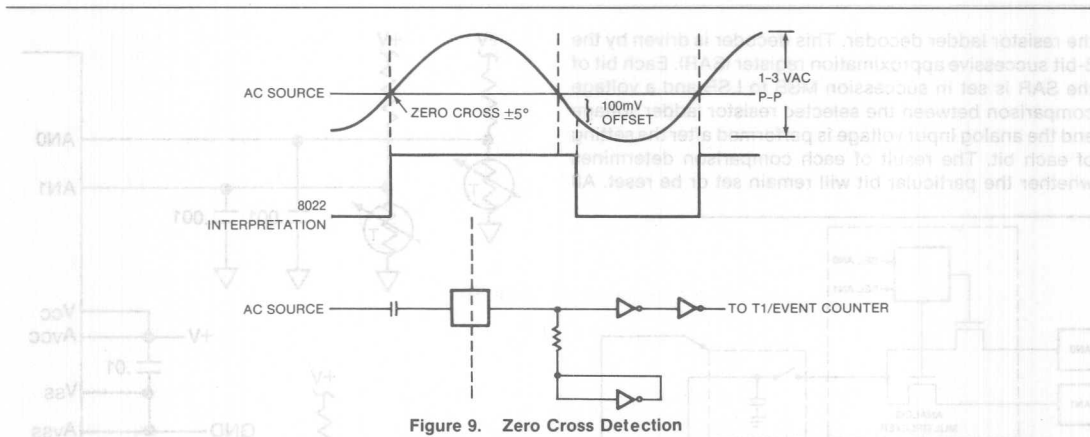


Figure 9. Zero Cross Detection

The phase angle at the T1 input can be expressed as

$$\theta = \arctan \frac{X_C}{R}$$

$$\text{where } X_C = \frac{1}{2\pi fC}$$

$$R = 150K\Omega \text{ (see fig. 10)}$$

Solving the equation using the recommended one microfarad capacitor and 60Hz

$$X_C = \frac{1}{2\pi (60) (1\mu f)}$$

$$= 2652.6$$

$$\theta = \arctan \frac{2652.6}{150K\Omega}$$

$$= -1.010$$

shows the voltage at the pin slightly leading the true AC voltage. Internally the circuit adds up to another five degrees before the processor can detect that a zero crossing occurred. Software can also add several degrees before outputting a signal. To compensate for all of this delay, a smaller capacitor could be chosen to give a -5 degree shift in hardware before the processor.

The zero cross detection capability allows the user to make the 50/60 Hz power signal the basis for his system timing. All timing routines, including time-of-day, can be implemented using the zero cross detection capability of T1 and its conditional jump instructions. In addition, the zero cross detection feature can be used in conjunction with the timer interrupt, as discussed earlier, to interrupt processing at the zero voltage point. This enables the user to control voltage phase sensitive devices such as triacs and SCRs, and to use the 8022 in applications such as shaft speed and angle measurement.

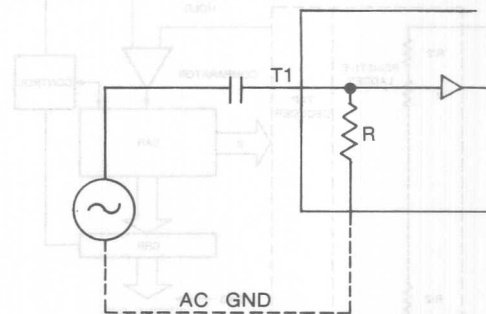


Figure 10. AC Equivalent of Zero Cross Input

Analog To Digital Converter

The T1 zero cross function is only one of the linear functions incorporated into the 8022 architecture. The most noted linear function is that of a complete analog to digital converter.

The analog to digital converter is a complete successive approximation converter with two multiplexed input channels. Either channel is selected by software with the SEL AN0 or SEL AN1 instruction. These instructions also restart the conversion sequences. A valid digital value can be read with the RAD (read A/D) instruction during the fourth instruction cycle following a select instruction. Conversions occur continuously, and RAD may be executed at any time with confidence that the sample is no more than 40 microseconds old.

The converter hardware has three parts as shown in Figure 11, a series string of resistors, a voltage comparator, and successive approximation logic. A series string of 256 matched resistors divides the voltage between AVSS and VAREF (the reference pin) into 256 voltage steps. This configuration gives the converter its inherent monotonicity.

The voltage tap on the series resistor string is selected by

the resistor ladder decoder. This decoder is driven by the 8-bit successive approximation register (SAR). Each bit of the SAR is set in succession MSB to LSB and a voltage comparison between the selected resistor ladder voltage and the analog input voltage is performed after the setting of each bit. The result of each comparison determines whether the particular bit will remain set or be reset. All

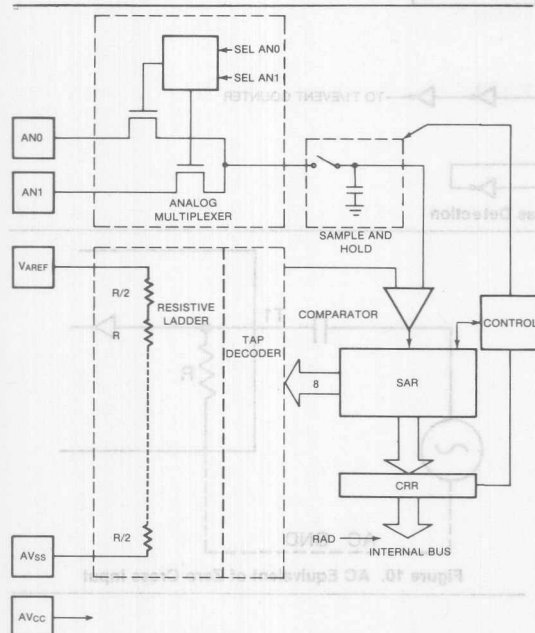


Figure 11. Analog to Digital Converter Block Diagram

comparisons are performed automatically by the on-chip A/D hardware. At the end of eight comparisons the SAR contains a valid digital representation of the analog voltage. This result is then latched into the conversion result register (CRR). The RAD instruction can then load the conversion result from the CRR to the accumulator.

To insure maximum accuracy from the A/D converter, separate power supply pins (Avcc and Avss) and a substrate pin (SUBST) have been provided. Unless there is excessive noise on the digital power supply, both Vcc and Avcc can be tied together and still maintain maximum accuracy. Figure 12 shows a typical analog configuration for sensing temperature in two thermistors. The substrate has both low frequency and high frequency bypass for noise immunity. The power supply pins (Vcc, Avcc) are bypassed with a .01 microfarad capacitor close to the chip. All other analog signals are bypassed with .001 microfarad capacitors for added noise rejection. (See also Software Noise Rejection)

As figure 11 shows, VAREF is connected to the top of the resistive ladder. When the selected analog channel is equal to or greater than VAREF the conversion result will equal 255 decimal (FF hexadecimal). The VAREF voltage can be generated in a number of ways depending on the

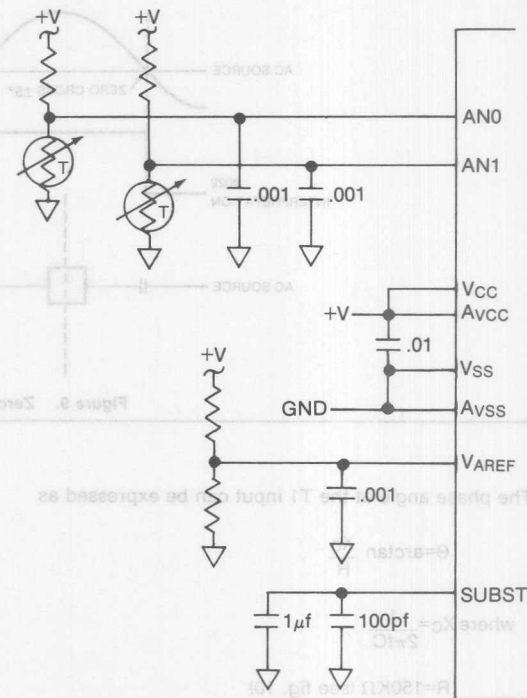


Figure 12. Typical Analog Schematic

system. It could be connected directly to Vcc giving a A/D range of GND to Vcc, or a simple resistor divider could be used to balance the reference voltage with the analog signals as in Figure 12. In calculating the impedance of the divider, the ladder impedance must be considered (see Figure 13). The total impedance of the ladder ranges from approximately 15K to 20K. This includes part to part differences and variance as a function of temperature. The resistor impedance should be chosen such that the 15K ohm parallel resistance is a small percentage of the divider impedance.

Input impedance of the converter can also be an important

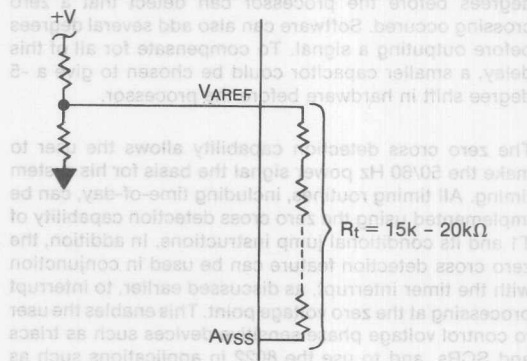


Figure 13. Ladder Impedance

factor. Figure 14 is an equivalent circuit of an analog input. Capacitance C1 is package capacitance which may range from 1pf to 3pf. Capacitance C2 is the sample and hold capacitance of 1.2pf to 1.4pf. This capacitance is only connected into the circuit by the sample and hold switch. The switch is closed for 0.3 tcy every four instruction cycles. Resistor R1 is package leakage which is approximately 2.5-5.0M ohms.

Software Noise Rejection



Figure 14. Analog Input Equivalent Circuit

Noise can be a problem in any system. Capacitors can be used to filter the noise but may not filter all of it. Capacitors also add cost to the system but can be eliminated by software filtering. One technique is simply to average two readings:

$$\frac{VIN1 + VIN2}{2} = VOUT$$

or keep a running average by averaging each reading with the previous average:

```
SEL AN0      ;Start conversion
MOV R0,#30   ;Point to storage location
RAD          ;Read current A/D sample
ADD A,@R0    ;Add current sample to previous average
RRC A        ;Divide by two
MOV @R0,A    ;Store new average
```

LOC	OBJ	LINE	SOURCE STATEMENT
		47 ;	
		48 ;	
		49 ;	AVERAGE 16 A/D READINGS
		50 ;	=====
		51 AVG16:	
002C	BC00	52	MOV R4,#00 ;CLEAR TEMP. MSB RESULT REGISTER
002E	B81A	53	MOV R0,#26 ;SET UP POINTER
0030	B000	54	MOV @R0,#00 ;CLEAR RESULT REGISTER
0032	85	55	SEL AN0 ;SELECT AND START CONVERSION
0033	BA10	56	MOV R2,#16 ;16 READINGS
		57	LOOP:
0035	80	58	RAD ;READ RESULT
0036	60	59	ADD A,@R0 ;LSB
0037	A0	60	MOV @R0,A ;SAVE NEW LSB
0038	27	61	CLR A
0039	7C	62	ADDC A,R4 ;ADD CARRY TO MSB
003A	AC	63	MOV R4,A ;SAVE NEW MSB
003B	EA35	64	DJNZ R2,LOOP ;NEXT READING
003D	47	65	SWAP A ;MSB INTO MSN
003E	20	66	XCH A,@R0
003F	47	67	SWAP A ;DIVIDE BY 16
0040	30	68	MOV @R0,A ;LOCATION 26 NOW CONTAINS
		69	THE AVERAGE
		70	
		71	
		72	

Figure 15.

This method will eliminate small fluctuations in the input voltage and reduce the effect of large fluctuations. Often, however, noise may be more severe. Excessive noise may require averaging of many readings taken over a short period of time.

$$\frac{VIN1 + VIN2 + \dots + VIN16}{16} = VOUT$$

Figure 15 lists the software required to average 16 successive A/D samples, as the above equation suggests. In such averaging, it is necessary to select the appropriate channel only once. Thereafter, a new conversion result is available every four instruction cycles.

Still another type of filtering is "exponential averaging." Similar to the running average method, current readings are averaged with the previous average.

$$\frac{VIN - Voldavg}{K} + Voldavg = Vavg$$

Where Vavg = current average
Voldavg = previous average
Vin = current reading
K = constant

This method has the advantage of large signal to noise ratios, but has slower dynamic response. In many systems, especially those involving temperature measurement, dynamic response is not a problem. Signal noise will be of a much higher frequency than any change in temperature. The constant, K, can be chosen to yield any desired signal to noise ratio. The larger the constant, the higher the ratio. The lower the constant, the higher the dynamic response.

To increase the effectiveness in reducing line generated noise, any of the above methods should be synchronized to the line frequency. As previously discussed, an interrupt can be generated when the 50Hz or 60Hz line frequency crosses AC zero. The A/D filtering routine should be part of the interrupt routine. Reading of the A/D will then occur at the same point of each line cycle, thus ignoring any line generated fluctuations in the analog inputs.

parators for easing the interface to non-digital inputs.

Port 0 has been modified from the standard quasi-bidirectional structure to allow an optional open drain configuration with comparator inputs. The low impedance pullup device has been eliminated and the high impedance pullup is optional. Thus, the user can choose via a mask programmable selection each line of Port 0 to be either quasi-bidirectional with a high impedance or true open-drain. The open drain configuration allows the line to sink current through the low impedance pulldown device or to float in the high output state. More importantly, the open drain configuration makes Port 0 very easy to drive when it is used as inputs. The input circuitry for each line of Port 0 includes a voltage comparator which amplifies the voltage difference between the input port line and the Port 0 threshold reference pin (V_{TH}). The voltage gain of the comparator is sufficient to sense a 100mV input differential within the range V_{SS} to $V_{CC}/2$.

If V_{TH} is allowed to float, it will bias itself to the digital switch point of the other ports, and Port 0 behaves as a set of normal digital inputs. However, by biasing V_{TH} , the switch point can be both tightly controlled and adjusted.

A typical use for Port 0 is in the interfacing with capacitive touch panels on microwave ovens and other new appliances. A touch-panel switch consists of two capacitors in series. One lead is attached to a high voltage buffer (10 to 30 volts). The other is attached to the Port 0 sense input. As a finger touches the common point, the drive signal is shunted by body capacitance, attenuating the signal reaching the input.

Low-voltage touch-panel operation (less than 30V) is possible, since the comparators allow small voltage changes to be detected. Most of the present touch-panel designs require a 50-100V drive on the touch panel.

Capacitive touch panels can be multiplexed in the same manner as mechanical keyboards (Fig. 16). The vacuum fluorescent display and the touch panel drivers are integrated to optimize hardware through shared high voltage buffers.

Figure 17 lists the software necessary to refresh the display and scan the touch-panel matrix. This routine could be adapted to serve as part of a timer/interrupt

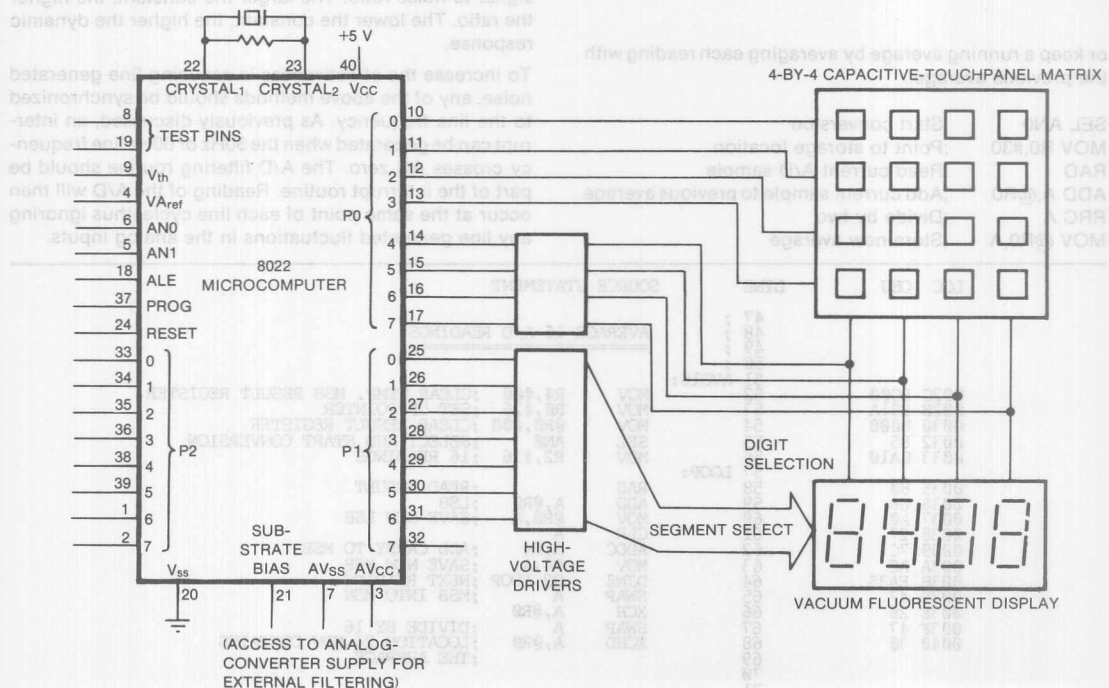


Figure 16. Typical Keyboard/Display Schematic

LOC	OBJ	LINE	SOURCE STATEMENT
		74 ;	
		75 ;	KEYBOARD DISPLAY ROUTINE
		76 ;	=====
	003C	77 D4	EQU 3CH ;MSD OF DISPLAY
		78	KEYDIS:
0041	27	80	CLR A
0042	39	81	OUTL P1,A ;TURN OFF SEGMENT DRIVERS
0043	90	82	OUTL P0,A ;TURN OFF DIGIT DRIVERS AND
		83	PANEL STROBES
		84	INITIALIZE SENSE INPUTS TO GND
0044	230F	85	MOV A,#0FH
0046	90	86	OUTL P0,A ;FLOAT SENSE INPUTS
0047	4B	87	ORL A,R3 ;NEW STROBE POSITION
0048	90	88	OUTL P0,A ;TURN ON STROBE
0049	08	89	IN A,P0 ;READ SENSE INPUTS
004A	AC	90	MOV R4,A ;SAVE SENSE INPUTS
004B	B83B	91	MOV R0,#D4-1 ;RAM LOCATION OF MSB OF 7-SEG PATTERN
004D	FB	92	MOV A,R3 ;STROBE POSITION INTO A
004E	90	93	OUTL P0,A ;GND SENSE INPUTS
		94	LOOP1:
004F	F7	95	RLC A ;ROTATE DIGIT STROBE INTO CARRY
0050	18	96	INC R0 ;NEXT DIGIT LOCATION
0051	E64F	97	JNC LOOP1 ;LOOP UNTIL CARRY
0053	F0	98	MOV A,@R0 ;RETRIEVE PATTERN FROM RAM
0054	39	99	OUTL P1,A ;OUTPUT NEW PATTERN
		100	
		101	

Figure 17.

scheme that would generate an interrupt at precise intervals for a flicker-free display. Another portion of the software would check for any touched input pads, test for valid entry, and enter key-depression codes into the main program.

Correcting Pad Imbalance

A common problem with capacitive touch panels is their imbalance. Layout, process, aging, and surface impurities all cause the capacitance to vary from touch pad to touch pad, resulting in a family of curves (Fig. 18) of voltage levels from each column of touch pads reaching the sense inputs. As the curves show, if threshold voltage V_{th1} alone were used, one column would always appear touched; if V_{th2} were used exclusively three of the columns would never appear touched.

To compensate for such varying capacitance levels, the on-chip analog input circuit may be used to allow multiple input voltage levels. Figure 19 depicts the 8022 version of such a circuit. $AN0$ and V_{TH} are tied together with a capacitor to line 0 of Port 0. The pull-up resistor and capacitor are used on line 0 to provide an RC timing network connected to $AN0$, V_{TH} , and $P00$. The remaining seven lines of Port 0 are the sense lines for the touch panel and use the open drain output option.

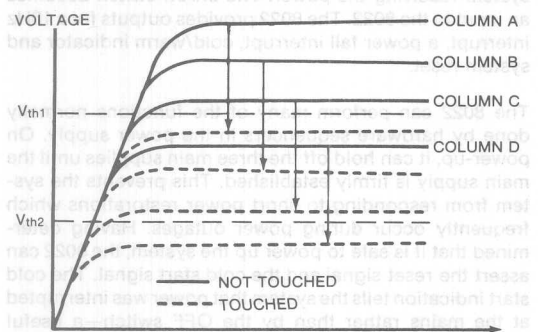


Figure 18.

All mnemonics copyrighted © Intel Corporation 1976.

Handling the different voltage levels would begin with initializing the data by plotting the family of curves with the $AN0$ input. This is done by writing a 0 to $P00$ (grounding $P00$) which initializes V_{TH} to 0v. A logic 1 is then written to $P00$, which begins to pull the RC network toward 5 v. As V_{TH} ramps upward, the sense lines are monitored for input changes, which will go from 1 to 0 as the not-touched voltage curve for each input is intersected. As the changes occur, the A/D value for each input is intersected. As the changes occur, the A/D value for each sense line can be read and stored. Thus the threshold reference voltage for each sense line can be determined by establishing the not touched voltage levels and by placing the threshold reference voltage below this level. As each row of the keyboard is scanned, the RC network is initialized to 0 v and ramps upward, varying the V_{TH} level. The A/D converter monitors this level looking for the calculated threshold points. As

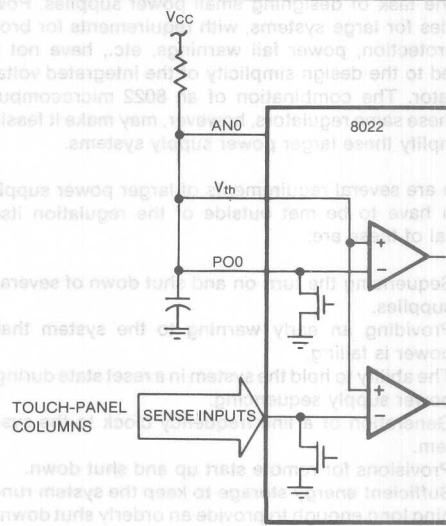


Figure 19.

the points are intersected, the corresponding sense inputs are read, with a 0 indicating a touched input and a 1 indicating a not-touched input. Thus, a multiplexed capacitive touch panel can be scanned and balanced without adding external components to the inputs.

For systems requiring more than two analog inputs to the 8022, Port 0 comparator inputs may be reconfigured to permit formation of pseudo-analog inputs from variable threshold digital inputs. The hardware configuration can be identical to that of Fig. 19. In this scheme, sense inputs act as additional analog inputs of less accuracy than AN0 and AN1 (about 6 bits).

The sequence for implementing the extension is essentially the same found in the variable-threshold touch panel. As V_{TH} ramps upward, a Port 0 bit is monitored for a change from 1 to 0. At the change, AN0 is read corresponding to the value of the analog input into Port 0 (with some error due to the time lag, which can be subtracted). This configuration can be utilized when it is possible to trade off accuracy for such cost improvements: one analog input with 8-bit accuracy and seven analog inputs with 6-bit accuracy. Such may be the case in a range controller that monitors temperature in two ovens, a meat probe, and two of the four burners, all to an accuracy within 10°F.

Application Ideas

This section will discuss some possible applications of the 8022. These applications are discussed in general terms and are believed to be feasible applications of the 8022. None of these applications, however, have been built and checked out.

Power Supply Controller

The three terminal voltage regulator, with its built-in current limiting and overload protection, has vastly simplified the task of designing small power supplies. Power supplies for large systems, with requirements for brown out protection, power fail warnings, etc., have not yet yielded to the design simplicity of the integrated voltage regulator. The combination of an 8022 microcomputer and these same regulators, however, may make it feasible to simplify these larger power supply systems.

There are several requirements of larger power supplies which have to be met outside of the regulation itself. Typical of these are:

1. Sequencing the turn on and shut down of several supplies.
2. Providing an early warning to the system that power is failing.
3. The ability to hold the system in a reset state during power supply sequencing.
4. Generation of a line frequency clock to the system.
5. Provisions for remote start up and shut down.
6. Sufficient energy storage to keep the system running long enough to provide an orderly shut down.
7. High efficiencies to minimize power requirements and heat dissipation.

These requirements can be met by a combination of raw DC supply, multiple three-terminal regulators, and an 8022 microcomputer. Figure 20 shows a raw supply which is capable of generating DC voltages suitable for regulation to five, plus twelve, and minus twelve voltages. (These are arbitrary, but common voltages). In addition, a separate winding is provided which generates a five-volt supply which will be used to supply power to the 8022 itself. The normal rectifiers in the RAW5 and RAW12 supplies are replaced by silicon controlled rectifiers which will be phase angle controlled by the 8022.

Figure 21 shows the connections to the 8022. The RAW5 and RAW12 supplies are applied to simple voltage dividers which feed the analog inputs of the 8022. The signal

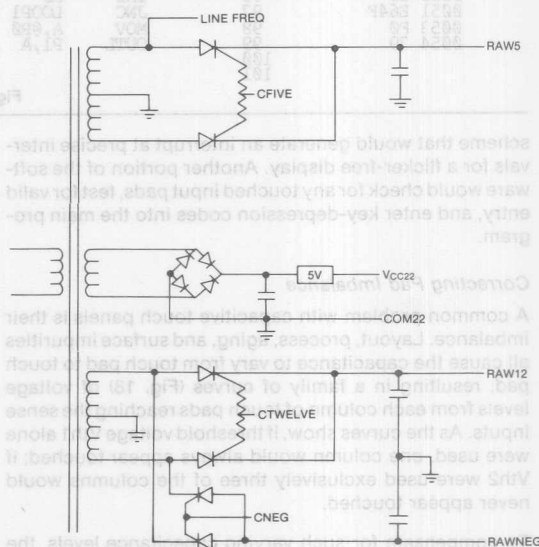


Figure 20.

LINEFREQ is taken from a convenient winding of a transformer, divided down, and applied to the zero cross input of the 8022. A strap is provided to configure the unit for 50/60 Hz operation. In addition to being connected to the basic power supply, the 8022 is also connected to the system receiving the power. The on/off switch becomes an input to the 8022. The 8022 provides outputs for a 10Hz interrupt, a power fail interrupt, cold/warm indicator and system reset.

The 8022 can perform many of the functions normally done by hardware sequencers in the power supply. On power-up, it can hold off the three main supplies until the main supply is firmly established. This prevents the system from responding to short power restorations which frequently occur during power outages. Having determined that it is safe to power up the system, the 8022 can assert the reset signal and the cold start signal. The cold start indication tells the system that power was interrupted at the mains rather than by the OFF switch—a useful function if any amount of battery backed up RAM exists in the system. Having set up these signals, the 8022 waits for

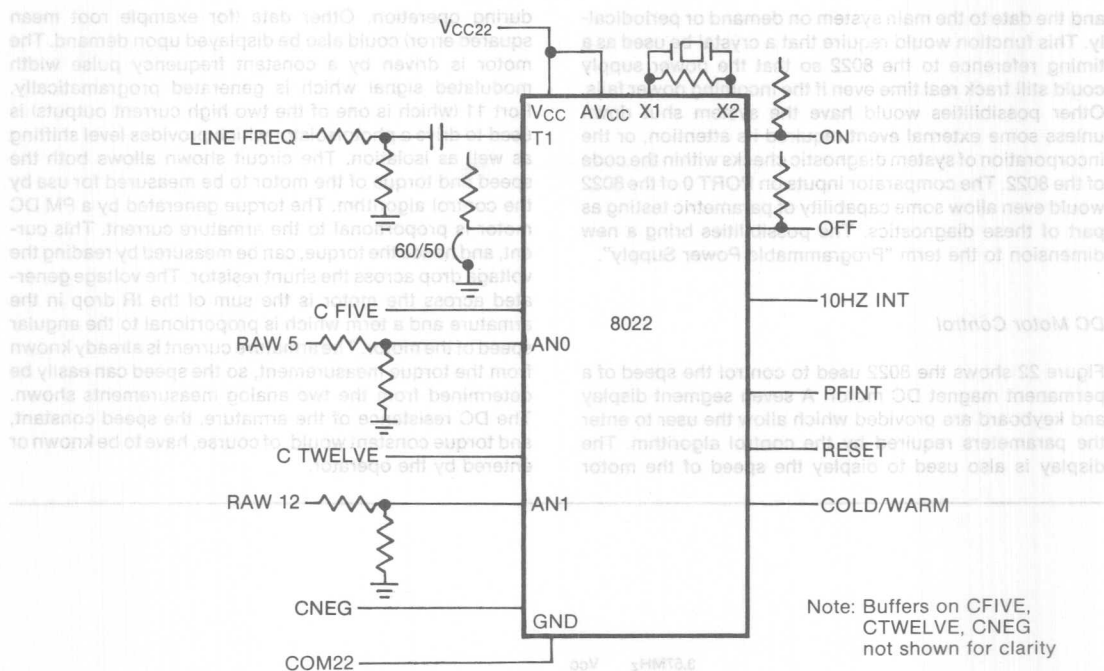


Figure 21.

a zero crossing (to minimize inrush) and then turns on the SCRs for the three supplies one at a time (again to minimize inrush). Any sequencing of the three supplies that is required by the system can also be allowed for. After some programmable time delay, the reset signal can be released and the system allowed to start operation.

During normal operation the 8022 can monitor the two major raw supplies and use phase angle control of the SCRs to regulate them. The regulation would be used to ensure that the three terminal regulators had minimum input voltage requirements met under all line voltage variations while at the same time minimizing the voltage drop across them. This increases the efficiency of the power supply and allows it to be capable of handling brown outs without dissipating excessive power in the regulators.

The line frequency input is used not only for the basis for the phase angle control, but also for two other functions; power fail detect and generation of the 10 Hz interrupt. The 10Hz interrupt can be generated by simply dividing the power line frequency by 5 for 50 Hz and 6 for 60 Hz operation. Performing this division in the power supply itself allows the system to be run on 50 or 60 cycle power with no change external to the power supply. In some situations it should even be possible to have the power supply adapt to either of these inputs by measuring the period of the incoming power on startup (see section "Which One?"). This would be an easy function to incorporate in the software and would require no additional hardware since provision is already made for zero cross detect.

Power fail detection can be done by running the timer while waiting for the line to zero cross. If an excessive time elapses it can be assumed that the power has failed and the power fail interrupt asserted. Note that this will detect total power failure but not a dip in the line voltage below the specifications of the power supply. This condition can be detected by keeping track of the phase angle that is required to maintain the RAW supplies at the proper level. If the SCR's have to be turned on for too high a portion of the total line cycle it is an indication of a brown-out condition and the powerfail interrupt should be generated. Whenever the powerfail interrupt is generated the 8022 should turn on the SCRs continuously to ensure maximum possible energy storage in the filter capacitors. After generation of the powerfail interrupt, the 8022 can again delay (depending, of course, on the energy storage of the power supply) and then assert reset. Once reset is asserted the SCRs are turned off, and left off, until the supplies have dropped down to a point which guarantees that any reset circuitry residing outside of the power supply will see a full power transition when power is reapplied. If the power is shut down by the 8022 in response to the on/off switch, the sequence would be similar except that the cold/warm start signal would indicate a warm start.

The above discussion should make it clear that the 8022 would make the task of designing a power supply system far easier, particularly for those designers more familiar with digital than analog design. If, in addition, the 8022 supply were put on a battery back-up, it would be possible to add many features to the system at virtually zero cost. The 8022 could be programmed to become the system clock and send, perhaps in serial ASCII, the time of day

Other possibilities would have the system shut down unless some external event required its attention, or the incorporation of system diagnostic checks within the code of the 8022. The comparator inputs on PORT 0 of the 8022 would even allow some capability of parametric testing as part of these diagnostics. The possibilities bring a new dimension to the term "Programmable Power Supply".

DC Motor Control

Figure 22 shows the 8022 used to control the speed of a permanent magnet DC motor. A seven segment display and keyboard are provided which allow the user to enter the parameters required by the control algorithm. The display is also used to display the speed of the motor

used to drive a photoisolator which provides level shifting as well as isolation. The circuit shown allows both the speed and torque of the motor to be measured for use by the control algorithm. The torque generated by a PM DC motor is proportional to the armature current. This current, and hence the torque, can be measured by reading the voltage drop across the shunt resistor. The voltage generated across the motor is the sum of the IR drop in the armature and a term which is proportional to the angular speed of the motor. The armature current is already known from the torque measurement, so the speed can easily be determined from the two analog measurements shown. The DC resistance of the armature, the speed constant, and torque constant would, of course, have to be known or entered by the operator.

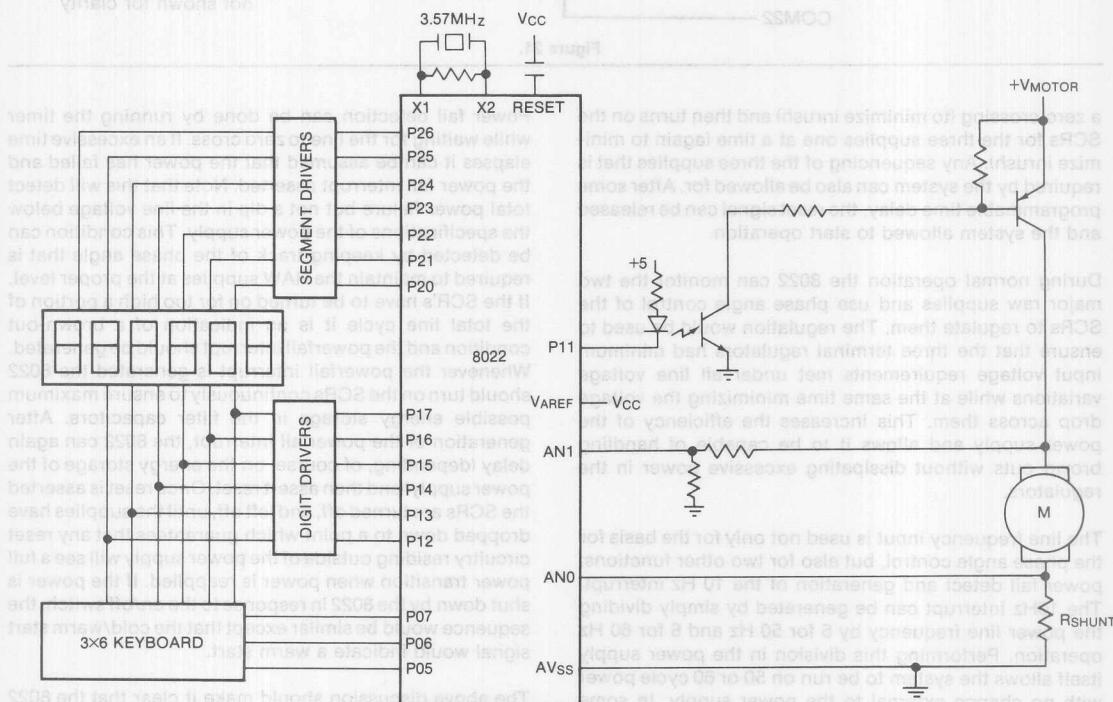


Figure 22. DC Motor Control

Figure 23 shows an automotive dashboard controlled by the 8022. Provisions are made to measure the oil pressure, water temperature, and vehicle speed. A connection to the distributor points allows the engine RPM and point dwell to be measured. Outputs are provided to control the ignition and starter (allowing the ignition switch to be eliminated in favor of a combination lock). Drive to a

vacuum servo is also possible to allow cruise control to be cheaply implemented. The display can be used for a speedometer, tachometer, oil pressure gauge, or water temperature gauge depending on the current desire of the driver. There are several uncommitted I/O pins which could be used to implement functions such as intermittent action windshield wipers or delayed action light circuits.

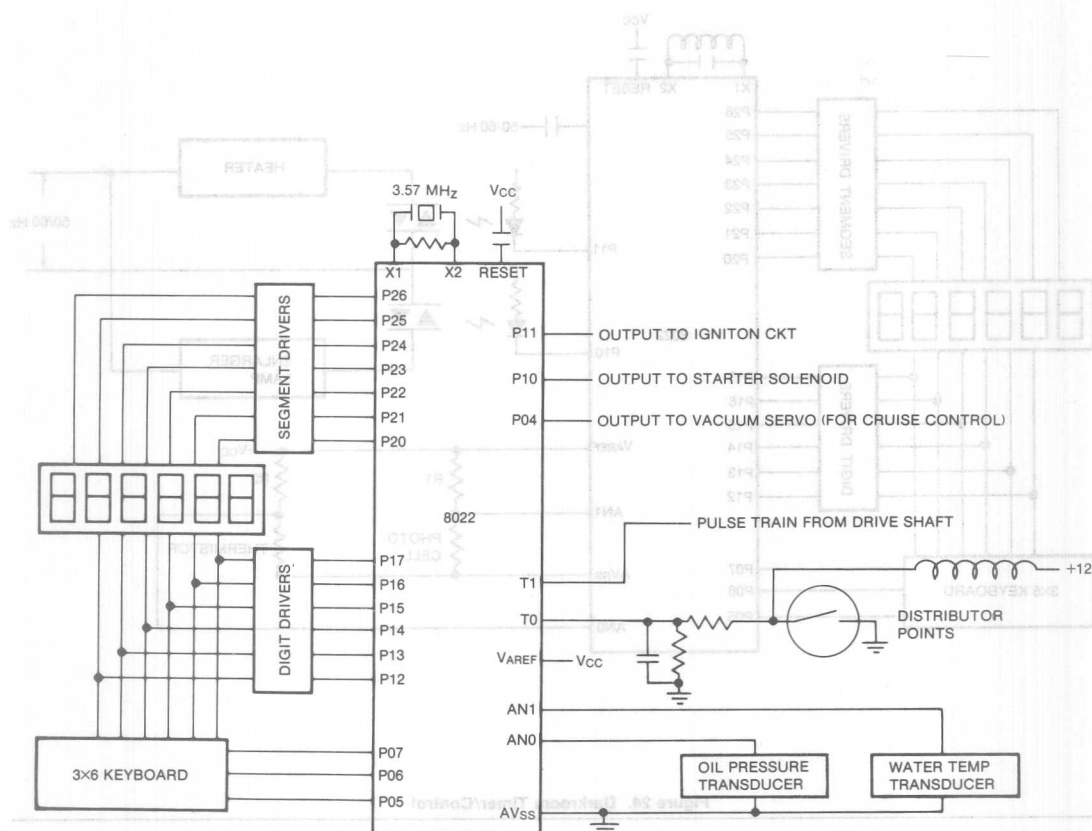


Figure 23. Automotive Dashboard

the enlarger, both of which can be controlled by the microcomputer. The 8022 could be used to run several timers concurrently while also maintaining the temperature of the chemical bath at the required level. Several uncommitted I/O pins are available for additional functions.



encountered in designing with the 8022.

The reader has also been exposed to several possible applications which show the versatility and cost effectiveness of a microcomputer with on-board analog features.

Energy Savings in an Induction Motor Using the 8022 Microcontroller

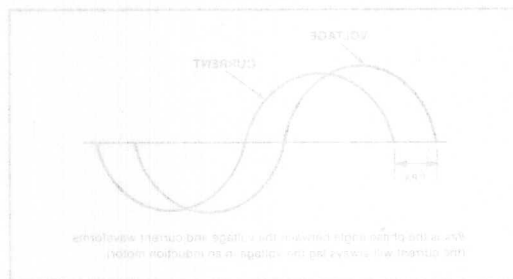


Figure 1.

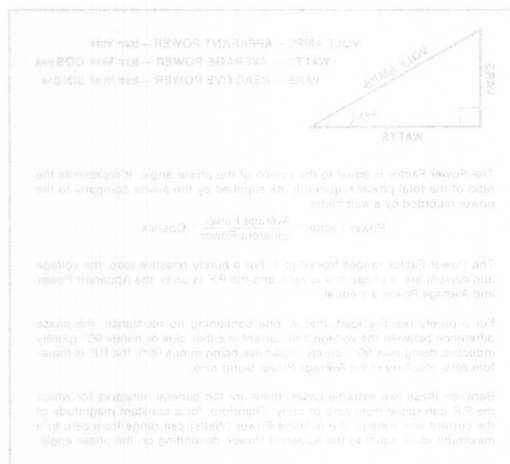


Figure 2. Complex Power and the Power Triangle

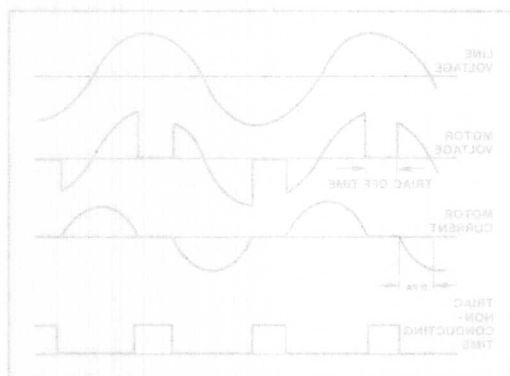


Figure 3.

Contents

INTRODUCTION	3-248
THEORY OF OPERATION	3-248
THE 8022	3-250
HARDWARE DESCRIPTION	3-251
SOFTWARE DESCRIPTION	3-251
CONCLUSION	3-253
PROGRAM LISTING	3-255

Since the induction motor is widely used in many of the same systems that can take advantage of the 8022 microcontroller, this invention can provide a significant benefit to many commercial and consumer products. Examples include the following:

- space heating and cooling systems
- heat pumps
- solar collector controllers
- liquid or chemical process control
- large industrial motor control
- refrigeration units
- swimming pool controllers
- washing machines
- dishwashers.

This application note will explain how this invention can be implemented with an 8022 microcontroller with a minimum amount of external hardware. The note is organized into the following sections:

1. Power Factor Controller theory of operation.
2. Description of the 8022 microcontroller in this application.
3. Hardware description.
4. Software description and listing.
5. Conclusion.

Theory of Operation

The concept of the Power Factor Controller (PFC) conceived and developed by NASA aerospace engineer Frank Nola, is to reduce the voltage applied to the motor when it is partially loaded, resulting in significant energy savings. At full load, the induction motor must have a high flux density in order to perform adequately. Under this condition the motor is running at peak efficiency with a power factor of about 0.8. At less than full load, however, this high flux density is still supported by the high current set up in the field coils. The resulting power factor can drop to as low as 0.1 (FIG. 2). The losses

NASA Tech Brief ME7-33280, Power Factor Controller. Copies may be obtained by writing to Director, Technology Utilization Office, Marshall Space Flight Center, AL 35812. The patent for the PFC is owned by the U.S. Government. Licenses for commercial development are available at no charge. Contact: Patent Counsel, Marshall Space Flight Center, AL 35812.

Introduction

A recent NASA invention enables considerable energy savings when using the common induction electric motor. It can be used with existing motors, since it requires no modification to the motor. Typical energy savings of 10 to 60% can be realized, depending on the amount of motor loading present. This invention is the direct result of an analysis, by NASA, of Solar Heating and Cooling Systems to reduce the power consumed by pump and fan motors used in these systems. It is applicable to both single phase and 3 phase motors.

Since the induction motor is widely used in many of the same systems that can take advantage of the 8022 microcontroller, this invention can provide a significant benefit to many commercial and consumer products. Examples include the following:

- space heating and cooling systems
- heat pumps
- solar collector controllers
- liquid or chemical process control
- large industrial motor control
- refrigeration units
- swimming pool controllers
- washing machines
- dishwashers.

This application note will explain how this invention can be implemented with an 8022 microcontroller with a minimum amount of external hardware. The note is organized into the following sections:

1. Power Factor Controller theory of operation.
2. Description of the 8022 microcontroller in this application.
3. Hardware description.
4. Software description and listing.
5. Conclusion.

Theory of Operation

The concept of the Power Factor Controller (PFC), conceived and developed by NASA aerospace engineer Frank Nola,¹ is to reduce the voltage applied to the motor when it is partially loaded, resulting in significant energy savings. At full load, the induction motor must have a high flux density in order to perform adequately. Under this condition the motor is running at peak efficiency with a power factor of about 0.8. At less than full load, however, this high flux density is still supported by the high current set up in the field coils. The resulting power factor can drop to as low as 0.1 (FIG. 2). The losses

¹ NASA Tech Brief MFS-23280, Power Factor Controller. Copies may be obtained by writing to: Director, Technology Utilization Office, Marshall Space Flight Center, AL, 35812. The patent for the PFC is owned by the U.S. Government. Licenses for commercial development are available at no charge. Contact: Patent Counsel, Marshall Space Flight Center, AL, 35812.

associated with this high current, under the lightly loaded condition, is primarily in the form of heat. This is a loss which can be paid for twice if the motor is in a refrigerated or otherwise cooled system.

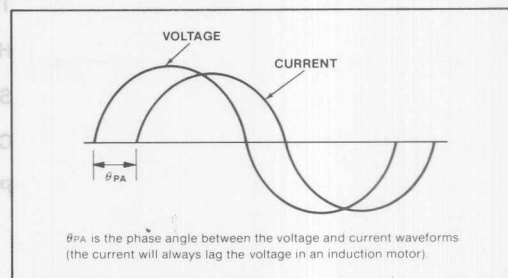


Figure 1.

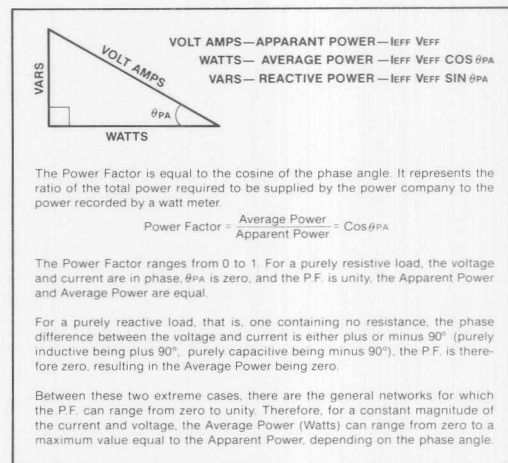


Figure 2. Complex Power and the Power Triangle

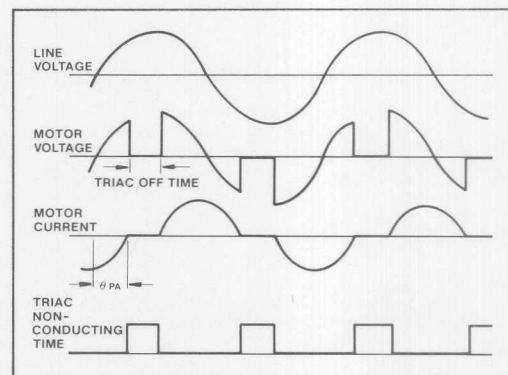


Figure 3.

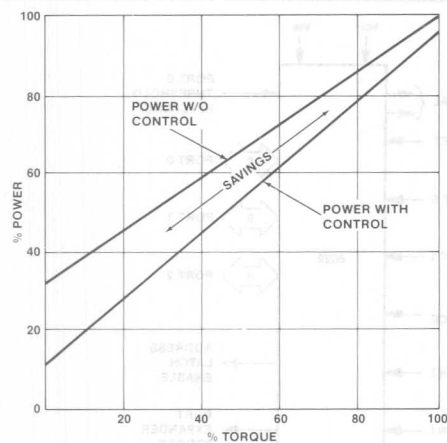


Figure 4. Typical Power Savings² for Single Phase Motor

In this partial load condition, the motor would produce the same torque at essentially the same speed even if much weaker magnetic forces were set up, by decreasing the current which produces these fields. The electric motor on its own, can't recognize this condition, however, and will continue to draw near the high current used under full load, even when operating under no load.

The principle of operation of the PFC is to measure the shift in phase angle between motor voltage and current

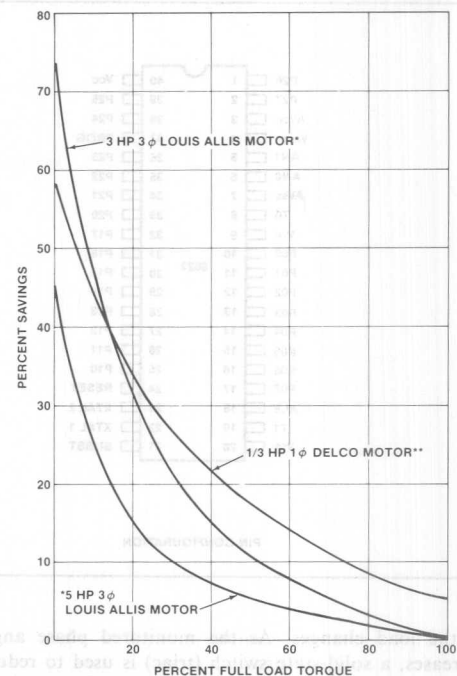


Figure 5. Power Factor Controller Percent Savings Vs. Torque for Various Motors³

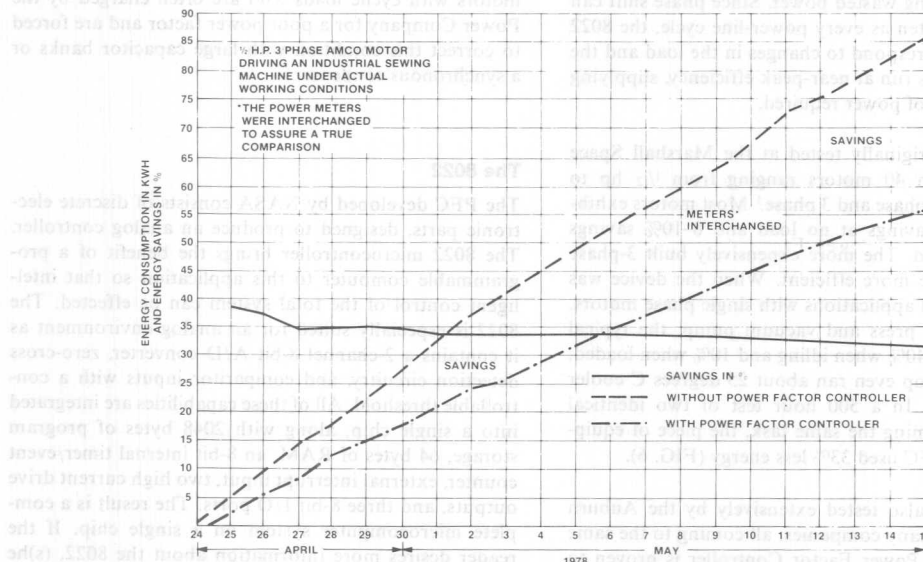


Figure 6. Energy Consumption as a Function of Time⁴

²NASA TECH BRIEF MFS-23280

³Ibid.

⁴Ibid.

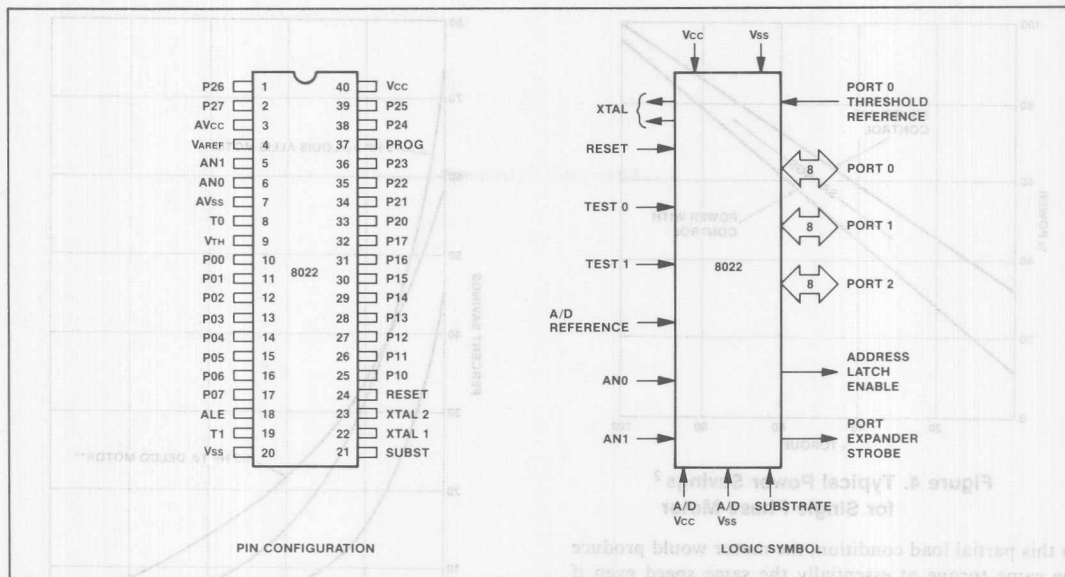


Figure 7.

as the load changes. As the monitored phase angle increases, a solid-state switch (triac) is used to reduce the voltage applied to the motor (FIG. 3). This increases motor slip and reduces the phase angle to a predetermined value. The feedback loop forces the motor to run at a constant, optimum, phase angle selected by the user, thereby minimizing wasted power. Since phase shift can be sampled as often as every power-line cycle, the 8022 can immediately respond to changes in the load and the motor will always run at near-peak efficiency, supplying only the amount of power required.

The PFC was originally tested at the Marshall Space Flight Center on 40 motors ranging from $1/12$ hp to 5 hp, both single phase and 3 phase.⁵ Most motors exhibited a 40-60% savings at no load and 0-10% savings when fully loaded. The more expensively built 3-phase motors being the more efficient. When the device was used in 15 typical applications with single phase motors, including a drill press and vacuum pump, the typical savings were 30-40% when idling and 10% when loaded. The vacuum pump even ran about 25 degrees C cooler with the device. In a 500 hour test of two identical machines performing the same task, the piece of equipment with the PFC used 33% less energy (FIG. 6).

The device was also tested extensively by the Auburn University and many companies, all coming to the same conclusion. The Power Factor Controller is proven to offer significant savings to the user that, by the way, go

beyond the actual direct power reduction mentioned. These multiple savings include the motor running cooler and quieter, extending the motor's life as well as saving in an air conditioned environment since there is less heat generated, and also by actually controlling the power factor to a desired value. This will benefit large users of motors with cyclic loads who are often charged by the Power Company for a poor power factor and are forced to correct this condition with large capacitor banks or a synchronous condenser.

The 8022

The PFC developed by NASA consists of discrete electronic parts, designed to produce an analog controller. The 8022 microcontroller brings the benefit of a programmable computer to this application so that intelligent control of the total system can be effected. The 8022 is especially suited for an analog environment as it contains a 2-channel 8-bit A/D converter, zero-cross detection circuitry, and comparator inputs with a controllable threshold. All of these capabilities are integrated into a single chip, along with 2048 bytes of program storage, 64 bytes of RAM, an 8-bit internal timer/event counter, external interrupt input, two high current drive outputs, and three 8-bit I/O ports. The result is a complete microcomputer system on a single chip. If the reader desires more information about the 8022, (s)he may refer to the MCS-48 User's Manual and application note AP-56, "Designing With Intel's 8022 Microcontroller," for a complete description of the 8022.

⁵Ibid.

The 8022 is already being used to control many systems which utilize an induction motor. The extra code required for the PFC operation is only 154 bytes (less than 8% of the available program store). The main program would, of course, have to be modified to facilitate the PFC's function. This can easily be performed by placing the PFC operation in an interrupt routine. The total number of pins delegated to the PFC operation is 4: three I/O pins and the T1 zero-cross input. One of these I/O pins is dedicated to control Vth.

These three interface lines: the voltage-cross input, the current-cross input, and the triac gate control, are all that is required for the PFC application. The 8022 simply measures the amount of lag time between the voltage zero-crossing and the current zero-crossing. This measured time is then converted to a phase angle and subsequently compared to the desired phase angle. As the load changes and the measured phase angle shifts either greater than or less than the desired value, the 8022 will either lengthen or shorten the triac off time. The result is that the motor only gets the amount of current needed to drive the instantaneous load.

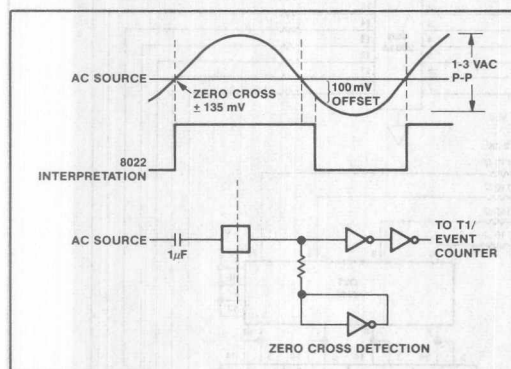


Figure 8.

Hardware Description

To perform the PFC function, the motor was placed in series with a triac which cuts out portions of the applied voltage, and a small series resistor which was used for measuring the current (FIG. 9). To isolate the 8022 from the AC line voltage, a small audio transformer was placed across the current sensing resistor, and an optoisolator was used between the gate of the triac and the pin driving it. The voltage signal was taken from the secondary side of the power supply transformer.

Aside from the isolation, the only other interface hardware was some simple signal conditioning. Since the T1 zero-cross input requires a 1-3 VAC p-p signal, two diodes to DC ground clipped the AC signal nicely

at 1.5 VAC p-p. A resistor for current limiting and a 1 microfarad capacitor to T1 complete the voltage zero-cross.

One of the port 0 comparator inputs was used for the current zero-cross detection. The 8022 had to be able to recognize when the current waveform was approaching zero from both the positive side and the negative side since it doesn't cross zero at any particular point, but rather approaches it from either side then remains zero until the triac fires again (FIG. 3). This was done by having two separate thresholds for the comparator. One for the negative slope crossing and one for the positive. To accomplish this, the current waveform was first boosted up with a DC level so it would be entirely positive. VTH was then biased to this DC level by two resistors. The threshold would be changed by about 0.1v either side of this DC level under software control by writing either a logic "1" or "0" to P17 when anticipating either a positive approach or a negative approach, respectively. This would place the 100k ohm resistor in parallel with either resistor thereby decreasing its value and subsequently raising or lowering the threshold.

The remaining hardware was included to aid in the development of this application note, but is not necessary for the system's operation. The remaining 6 pins on port zero were used to input the desired phase angle. The remaining 7 pins on port 1 were used to control the 7 segments of a display. The upper nibble of port 2 was used to control up to 10 digits of a display. The user would then select the desired phase angle at will, with the actual phase angle being written to the display.

With this pin assignment the remaining pins include the lower nibble of port 2 which can be used with the Intel 8243 I/O expander, adding four 4-bit I/O ports. The T0 pin can still be used as either a testable input, the interrupt request pin, or both. Finally, the two channels of the A/D converter are available, allowing the chip to interface directly with analog transducers. Aside from the Power Factor Controller and the direct user interface with the 8022 (via a keypad or thumbwheel switches and the display), there are still enough pins left over for almost any controller application.

Hardware not required for the PFC operation is shaded in the schematic.

Software Description

To simplify the software, the triac "offtime" is quantized in units of 0.27ms. This represents one time-unit of the timer (operating with a 3.58 MHz TV crystal). The "offtime" is therefore incremented or decremented by one of these units (or remained unchanged) when it is

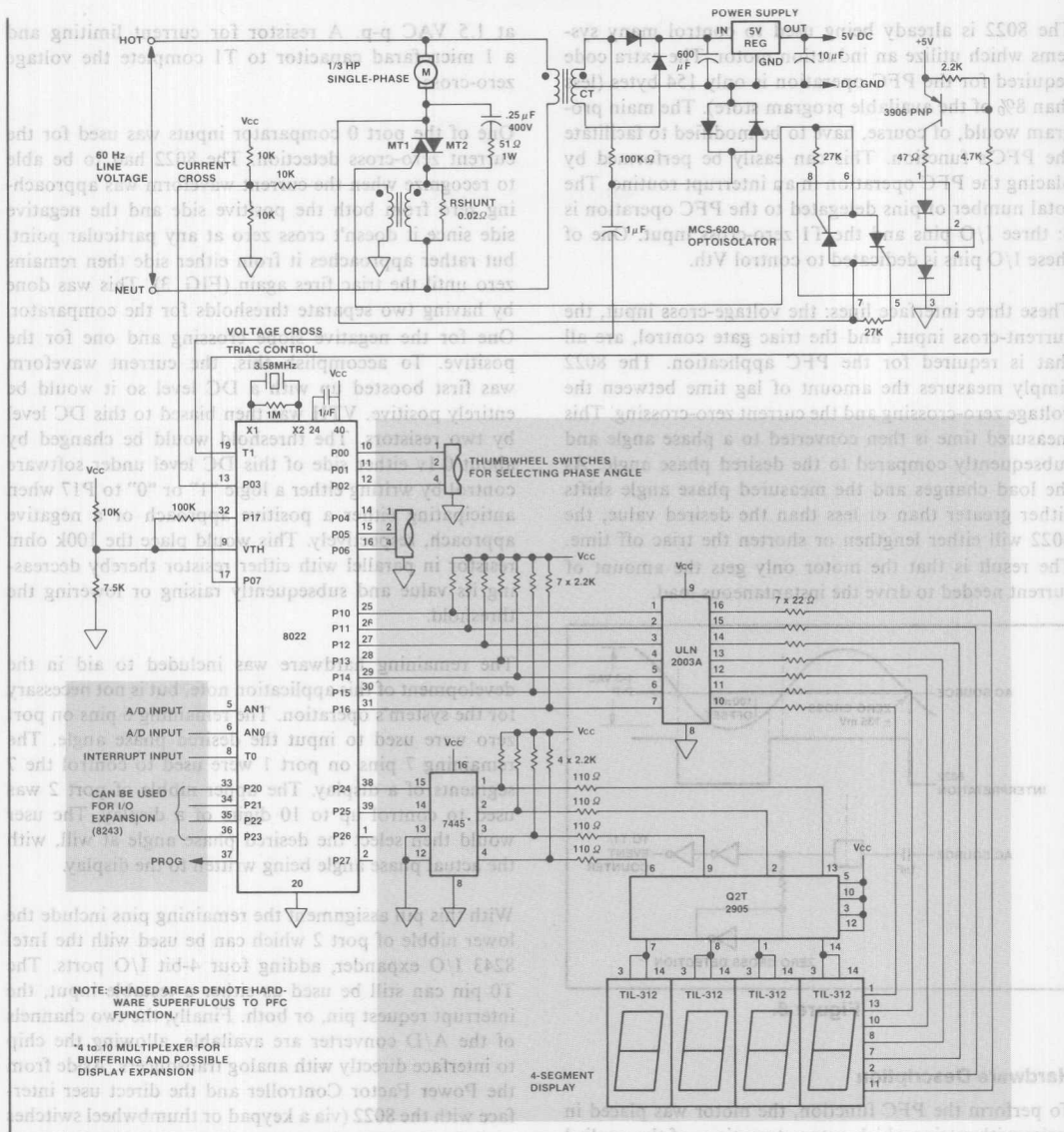


Figure 9.

reviewed for adjustment every cycle. This proves to be an adequate reaction to abrupt changes in the load. If however, the phase angle will only be sampled every 3 cycles, for example, on an interrupt, a simple routine has been included to change the "offtime" by more than one increment should the phase angle change more than 12 degrees between samplings. This is included to compensate for extremely abrupt load changes.

For simplicity, the phase angle is also measured in units of approximately 6 degrees, to coincide with the timer unit of 0.27ms. The lookup table consists of these quantized values. For a steady phase angle, the desired phase angle should be chosen to be one of these values. If any value between two of these are chosen, the phase angle oscillates between these two values and will never equal the chosen value, since this value is not in the lookup

table. This can be rectified by using a table lookup and interpolation instead of a simple table lookup (see AP-24; "Application Techniques for the MCS-48 Family"). There is no apparent benefit in doing this, however, as the value-searching over a 6 degree range doesn't seem to cause any problems.

The software also distinguishes between that necessary for the PFC only, and that used to include the user interface features. The BCD-to-binary routine was adapted from AP-49, "Serial I/O and Math Utilities for the 8049 Microcomputer," as was the binary-to-BCD routine. The total bytes of program storage consumed by the PFC only, is 154, and the total for this entire application is 328.

The software flowchart and complete listing follow.

Conclusion

The advantages of controlling a system with a programmable microcontroller are evident. The added advantage of conserving power with the 8022 in the same application is realized in this application note. As the Power Factor Controller will be used in more and more applications as an energy-saving feature, the 8022 is ideally suited to implement it.

To illustrate this, the following application shows the 8022 controlling a Heat Pump / Solar Heating system.

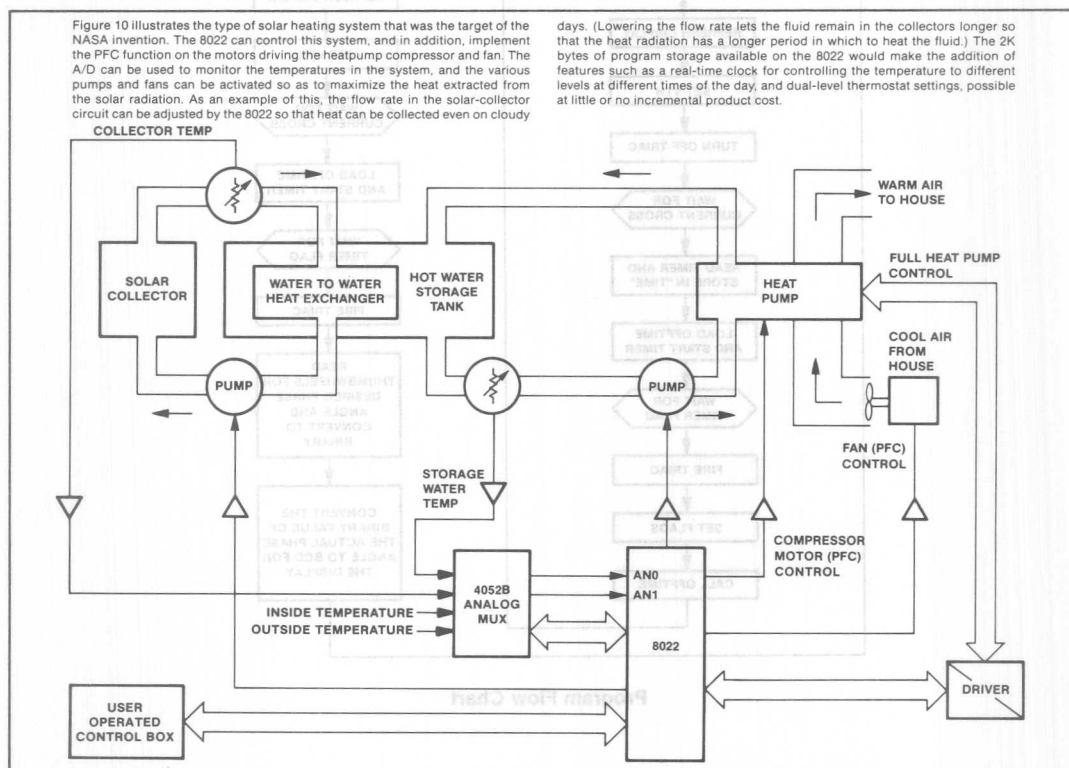


Figure 10.

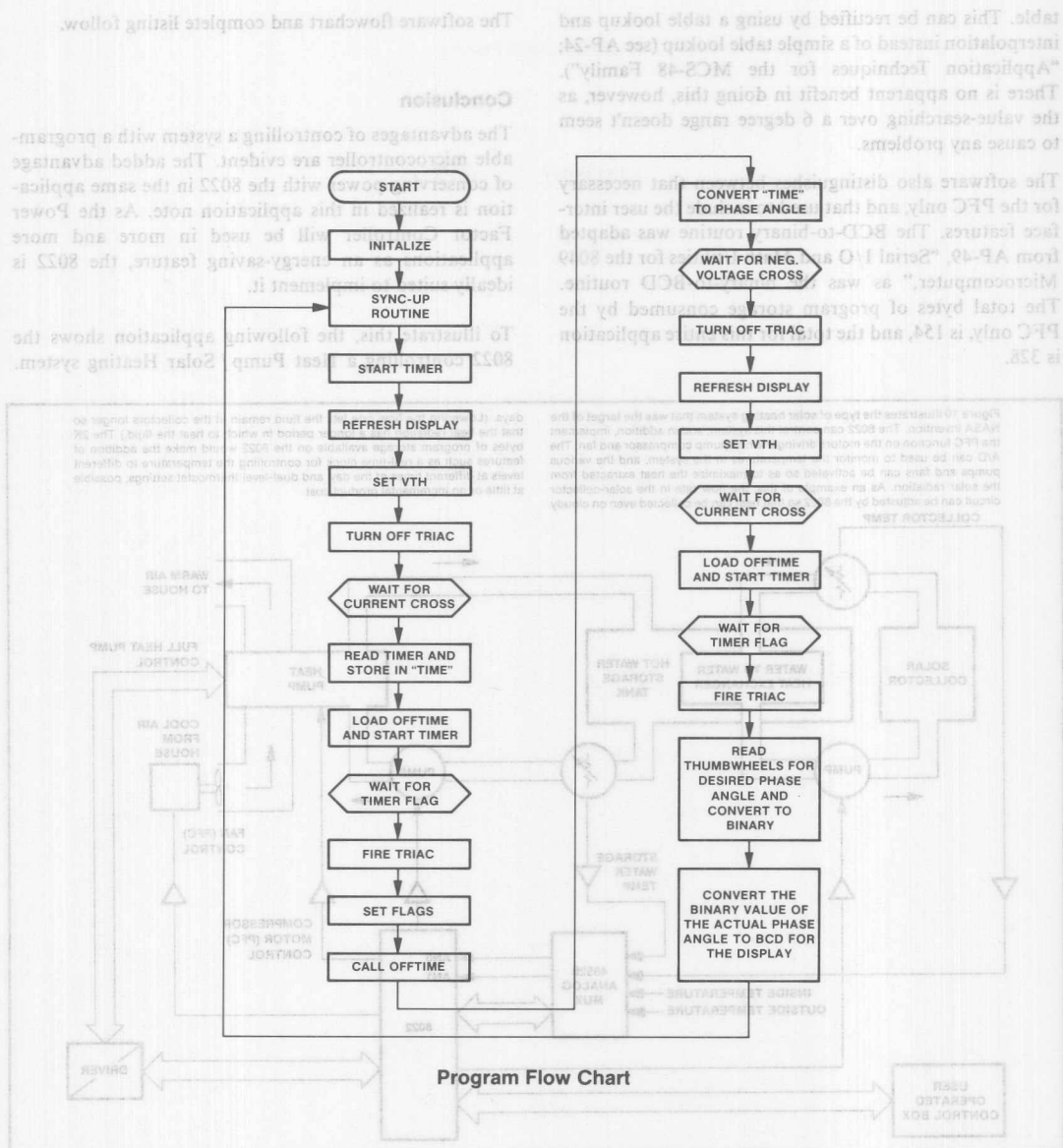


Figure 10.

```

LOC OBJ      SEQ      SOURCE STATEMENT
*****
1 $MOD22 SYMBOLS MACROFILE XREF
2 *****
3 *****
4 ***** DEFINITIONS OF MACROS *****
5 *****
6 *****
7 LDA        MACRO    PAR1
8 MOV        RO,#PAR1
9 MOV        A,@RO
10 ENDM
11 STA        MACRO    PAR1
12 MOV        RO,#PAR1
13 MOV        @RO,A
14 ENDM
15
16 0000 1498 INIT: CALL INITZ ;INITIALIZE
17 *****
18 ***** THE FOLLOWING IS A SYNC-UP ROUTINE TO ASSURE
19 ***** A START ON A RAISING VOLTAGE ZERO-CROSS *****
20 *****
21 *****
22 *****
23
24 0002 5606 PWRON: JT1; VOWAIT ;JUMP ON T1=1
25 0004 0402 JMP PWRON
26
27 0006 460A VOWAIT: JNT1; WAITST ;JUMP ON T1=0
28 0008 0406 JMP VOWAIT
29
30 000A 560E WAITST: JT1; START ;JUMP ON T1=1
31 000C 040A JMP WAITST
32 *****
33 ***** END OF SYNC-UP *****==START PROGRAM==*****
34 *****
35 *****
36
37 000E 27 START: CLR A
38 000F 62 MOV T,A ;CLEAR PRESCALER
39 0010 55 STRT T ;START TIMER
40
41 0011 3400 CALL REFRSH ;REFRESH THE DISPLAY
42 ;VTH SET FOR UPCOMING CURRENT AUTO-
43 ;MATICALLY IN REFRSH ROUTINE (P17=0)
44
45 0013 23FF MOV A,#OFFH
46 0015 90 OUTL PO,A ;TURN OFF TRIAC
47
48 0016 08 IPWAIT: IN A,PO
49 0017 5308 ANL A,#00001000B ;WAIT FOR CURRENT CROSS
50 0019 C616 JZ IPWAIT
51
52 001B 42 MOV A,T ;READ TIMER
53 001C AA MOV TIME,A ;STORE TIMER VALUE
54
55 001D 14BB CALL TRIAC
56
57 $EJECT

```

NOTE: SHADED AREAS DENOTE SOFTWARE SUPERFLUOUS TO PFC FUNCTION.

LOC	OBJ	SEQ	SOURCE STATEMENT
58			*****
59			*****
60			*****
61			*****
62			*****
63	001F 97		CLR C ;CLEAR THE CARRY FLAG
64			
65	0020 FC		MOV A,SETVAL
66	0021 37		CPL A
67	0022 17		INC A ;2'S COMPLIMENT SETVAL
68	0023 6D		ADD A,PHSANG ;PHSANG-SETVAL = DELTA
69			
70	0024 C64F		JZ LOOK ;IF PHSANG=SETVAL, DONT CHANGE ANYTHING
71			
72			STA DELTA
73	0026 B826		MOV RO,#DELTA
74	0028 A0		MOV @RO,A
75			
76	0029 5380		ANL A,#10000000B ;TEST FOR NEG
77	002B C638		JZ OTCALC ;IF POS DO NOT SET FLAG
78	002D 1E		INC PHLTSV ;SET PHSANG L.T. SETVAL FLAG
79			
80			LDA DELTA
81	002E B826		MOV RO,#DELTA
82	0030 F0		MOV A,@RO
83	0031 030A		ADD A,#10D ;10-DELTA
84	0033 5380		ANL A,#10000000B
85	0035 C638		JZ OTCALC
86	0037 1F		INC DLGT10 ;SET DELTA G.T. 10 FLAG IF NEG
87			
88			*****
89			*****
90			*****
91			*****
92	0038 FE		OTCALC: MOV A,PHLTSV ;DECREMENT OR INCREMENT?
93	0039 C649		JZ DECRM ;IF PHLTSV IS CLEAR THEN DECREMENT
94			
95	003B FB		INCREM: MOV A,OFFTIM ;IF OFFTIME=255 OR 254 DON'T INCREMENT
96	003C 37		CPL A
97	003D C64C		JZ CLRFLG
98	003F 07		DEC A
99	0040 C64C		JZ CLRFLG
100			
101	0042 1B		INC OFFTIM ;INCREMENT
102			
103	0043 FF		MOV A,DLGT10 ;SHOULD IT INCREMENT TWICE?
104	0044 C64C		JZ CLRFLG
105	0046 1B		INC OFFTIM
106	0047 044C		JMP CLRFLG
107			
108	0049 FB		DECRM: MOV A,OFFTIM
109	004A 07		DEC A
110	004B AB		MOV OFFTIM,A ;DECREMENT
111			
112	004C 27		CLRFLG: CLR A
113	004D AE		MOV PHLTSV,A
114	004E AF		MOV DLGT10,A ;CLEAR FLAGS
115			
116			\$EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT	THUMB STATEMENT	300002	032	080 001
		117					
		118					
		119		LOOKUP VALUE FOR "PHSANG" FROM "TIME"			
		120					
004F	23C9	121	LOOK:	MOV	A, # (LOW TAB)		
0051	6A	122		ADD	A, TIME		
0052	A3	123		MOVP	A, @A		
0053	AD	124		MOV	PHSANG, A		
		125					
		126					
		127					
		128					
		129					
		130					
0054	4658	131	HOLD:	JNT1	CONTON		
0056	0454	132		JMP	HOLD		
		133					
0058	23FF	134	CONTON:	MOV	A, #OFFH		
005A	90	135		OUTL	P0, A		
		136					
005B	3400	137		CALL	REFRSH		
		138		LDA	SEG DAT		
005D	B824	139+		MOV	RO, #SEG DAT		
005F	F0	140+		MOV	A, @RO		
0060	4380	141		ORL	A, #10000000B		
0062	39	142		OUTL	P1, A		
		143					
0063	08	144	INWAIT:	IN	A, P0		
0064	5308	145		ANL	A, #00001000B		
0066	9663	146		JNZ	INWAIT		
		147					
0068	14BB	148		CALL	TRIAC		
		149					
		150					
		151					
		152					
		153					
		154					
006A	08	155		IN	A, P0		
006B	5377	156		ANL	A, #01110111R		
		157		STA	PINTR		
006D	B820	158+		MOV	RO, #PINTR		
006F	A0	159+		MOV	@RO, A		
		160					
0070	14A7	161		CALL	SELRB1		
		162					
		163		LDA	PINTR		
0072	B820	164+		MOV	RO, #PINTR		
0074	F0	165+		MOV	A, @RO		
		166					
0075	344A	167		CALL	CONBIN		
		168					
		169		STA	PINTR		
0077	B820	170+		MOV	RO, #PINTR		
0079	A0	171+		MOV	@RO, A		
		172					
007A	14B1	173		CALL	SELRBO		
		174					
		175		LDA	PINTR		
007C	B820	176+		MOV	RO, #PINTR		
007E	F0	177+		MOV	A, @RO		
007F	AC	178		MOV	SETVAL, A		
		179					
		180					

180 \$EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT
		181	*****
		182	*****
		183	CONVERT BINARY VALUE OF THE PHASE ANGLE (PHSANG)
		184	TO BCD (ANGLE)
		185	*****
0080	FD	186	
		187	MOV A,PHSANG
		188	STA BCDTR ;STORE PHASE ANGLE
0081	B821	189+	MOV RO,#BCDTR
0083	A0	190+	MOV @RO,A
		191	
0084	14A7	192	CALL SELRB1
		193	
		194	LDA BCDTR
0086	B821	195+	MOV RO,#BCDTR
0088	FO	196+	MOV A,@RO
		197	
0089	3437	198	CALL CNBCD
		199	
		200	STA BCDTR
008B	B821	201+	MOV RO,#BCDTR
008D	A0	202+	MOV @RO,A
		203	
008E	14B1	204	CALL SELRBO
		205	
		206	LDA BCDTR
0090	B821	207+	MOV RO,#BCDTR
0092	FO	208+	MOV A,@RO
		209	STA ANGLE
0093	B823	210+	MOV RO,#ANGLE
0095	A0	211+	MOV @RO,A
		212	
		213	
0096	040A	214	JMP WAITST ;GO BACK TO BEGINNING AND REPEAT
		215	*****
		216	*****
		217	*****
		218	*****
		219	----- END OF MAIN PROGRAM -----
		220	*****
		221	*****
		222	*****
		223	
		224	\$EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT
		225	INITZ:
		226	*****
		227	*****
		228	*****
		229	*****
		230	*****
		231	*****
0020		232	RINTR EQU 20H
0021		233	BCDTR EQU 21H
0022		234	PASTOR EQU 22H
0023		235	ANGLE EQU 23H
0024		236	SEG DAT EQU 24H
0025		237	DISFLG EQU 25H
0026		238	DELTA EQU 26H
		239	*****
0002		240	TIME EQU R2
0003		241	OFFTIM EQU R3
0004		242	SETVAL EQU R4
0005		243	PHSANG EQU R5
0006		244	PHLTSV EQU R6
0007		245	DLGT10 EQU R7
		246	*****
		247	*****
		248	FOR USE IN CONBIN AND CONBCD
		249	*****
		250	*****
0002		251	XA EQU R2
0003		252	COUNT EQU R3
0004		253	ICNT EQU R4
0001		254	DIGPR EQU 1
		255	*****
		256	*****
0098 BC29		258	MOV R4, #29H ;SETVAL=41
009A BD29		259	MOV R5, #29H ;PHSANG=41
009C BBFF		260	MOV R3, #0FFH ;SET OFFTIM TO 255
		261	*****
009E 237F		262	MOV A, #7FH ;TURN ON TRIAC
00A0 90		263	OUTL PO, A
		264	*****
00A1 97		265	CLR C
00A2 27		266	CLR A
		267	STA DISFLG ;CLEAR DISPLAY FLAG
00A3 B825		268+	MOV RO, #DISFLG
00A5 A0		269+	MOV @RO, A
		270	*****
00A6 83		271	RET
		272	*****
		273	\$EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT
274			*****
275			THESE NEXT TWO SUBROUTINES ACT LIKE A BANK SELECT
276			EXCEPT THAT R0 AND R1 GET WIPED OUT
277			*****
278			
279			
280			
281			
282			
283			
284			
285			SELRRP1:
286			MOV R1,#18H
287	00A7 B918		MOV R0,#07H
288	00A9 B807		MOV A,@R0
289	00AB F0		MOV @R1,A
290	00AC A1		INC R1
291	00AD 19		DJNZ R0,BAK1
292	00AE E8AB		RET
293	00B0 83		
294			
295			*****
296			
297			SELRR0:
298			MOV R1,#18H
299			MOV R0,#07H
300	00B1 B918		MOV A,@R1
301	00B3 B807		MOV @R0,A
302	00B5 F1		INC R1
303	00B6 A0		DJNZ R0,PAK2
304	00B7 19		RET
305	00B8 E8B5		
306	00BA 83		
307			\$EJECT

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

PAGE 7

LOC	OBJ	SEQ	SOURCE STATEMENT
307			*****
308			THIS ROUTINE FIRES THE TRIAC AFTER A
309			PREDETERMINED OFFTIME
310			*****
311			
312			
313			
314			
315			TRIAC:
316			
317			
318	00BB 65		STOP TCNT ;STOP TIMER
319	00BC 16BE		JTF \$+2 ;CLEAR TIMER FLAG
320			
321	00BE FB		MOV A,OFFTIM ;SET PRESCALER TO OFFTIME
322	00BF 62		MOV T,A
323			
324	00C0 55		STRT T ;START TIMER
325			
326	00C1 16C5		CNTDWN: JTF FIRE ;WAIT FOR TIMER FLAG
327	00C3 04C1		JMP CNTDWN
328			
329	00C5 237F		FIRE: MOV A,#7FH ;FIRE
330	00C7 90		OUTL PO,A ;TRIAC
331			
332	00C8 83		RET
333			
334	00C9		TAB EQU \$
335			
336			\$EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT	STATEMENT	LOC	OBJ
		337	*****			
		338	*****			
		339	THIS ROUTINE REFRESHES THE DISPLAY			
		340	*****			
		341	*****			
0100		342	ORG 100H			
		343	REFRSH:			
0100 23F0		344	MOV A,#0F0H			
0102 3A		345	OUTL P2,A	;TURN OFF DIGITS		
		346				
		347	LDA DISFLG			
0103 B825		348+	MOV RO,#DISFLG			
0105 F0		349+	MOV A,@RO			
0106 C60E		350	JZ NOSWAP	;CHECK DISPLAY FLAG		
		351	LDA ANGLE			
0108 B823		352+	MOV RO,#ANGLE			
010A F0		353+	MOV A,@RO			
010R 47		354	SWAP A			
010C 2411		355	JMP CONT2			
		356				
		357	NOSWAP: LDA ANGLE			
010E B823		358+	MOV RO,#ANGLE			
0110 F0		359+	MOV A,@RO			
0111 530F		360	CONT2: ANL A,#0FH	;MASK UPPER NIBBLE		
0113 032D		361	ADD A,#(LOW TABL)			
0115 A3		362	MOVP A,@A	;GET CHAR CODE		
		363				
		364	STA SEG DAT			
0116 B824		365+	MOV RO,#SEG DAT			
0118 A0		366+	MOV @RO,A			
0119 39		367	OUTL P1,A	;TURN ON THE SEGMENTS		
		368				
		369	LDA DISFLG			
011A B825		370+	MOV RO,#DISFLG			
011C F0		371+	MOV A,@RO			
011D C623		372	JZ DIG1			
		373				
011F 2310		374	DIG2: MOV A,#10H			
0121 2424		375	JMP CONT3			
		376				
0123 27		377	DIG1: CLR A			
		378				
0124 3A		379	CONT3: OUTL P2,A	;TURN ON THE DIGIT		
		380				
		381	LDA DISFLG			
0125 B825		382+	MOV RO,#DISFLG			
0127 F0		383+	MOV A,@RO			
0128 37		384	CPL A			
		385	STA DISFLG	;CHANGE DISPLAY FLAG		
0129 B825		386+	MOV RO,#DISFLG			
012B A0		387+	MOV @RO,A			
		388				
012C 83		389	RET			
		390				
		391	TABL:			
012D 3F		392	DB 00111111B	: 0		
012E 06		393	DB 00000110B	: 1		
012F 5B		394	DB 01011011B	: 2		
0130 4F		395	DB 01001111B	: 3		
0131 66		396	DB 01100110B	: 4		
0132 6D		397	DB 01101101B	: 5		
0133 7D		398	DB 01111101B	: 6		
0134 07		399	DB 00000111B	: 7		
0135 7F		400	DB 01111111B	: 8		
0136 67		401	DB 01100111B	: 9		
		402				
0137		403	HERE1 EQU \$			
		404				
		405	\$EJECT			

LOC	OBJ	SEQ	SOURCE STATEMENT	MEMORY ADDRESS	LOC	OBJ	SEQ	SOURCE STATEMENT	MEMORY ADDRESS
		406	*****				407	*****	
		408	LOOKUP TABLE FOR THE PHASE ANGLE				409	*****	
		410	*****				411	*****	
00C9		412					413	ORG	TAB
00CA 00		414	DB	00H			415	DB	00H
00CB 06		416	DB	06H			417	DB	0CH
00CC 0C		418	DB	11H			419	DB	17H
00CD 17		420	DB	1DH			421	DB	23H
00CE 1D		422	DB	29H			423	DB	2EH
00CF 23		424	DB	34H			425	DB	3AH
00D0 29		426	DB	40H			427	DB	46H
00D1 2E		428	DB	4BH			429	DB	51H
00D2 34		430	DB	57H			431	DB	5DH
00D3 3A		432	DB	63H			433	DB	68H
00D4 40		434	DB	6EH			435	DB	74H
00D5 46		436	DB	7AH			437	DB	80H
00D6 4B		438	DB	85H			439	DB	8BH
00D7 51		440	DB	91H			441	DB	97H
00D8 57		442	DB	97H			443	DB	97H
00D9 5D		444	DB	97H			445		
00DA 63		446	\$EJECT						
00DB 68									
00DC 6E									
00DD 74									
00DE 7A									
00DF 80									
00E0 85									
00E1 8B									
00E2 91									
00E3 97									
00E4 97									
00E5 97									
00E6 97									

LOC	OBJ	SEQ	SOURCE STATEMENT
		447	
0137		448	ORG HERE1
		449	
		450	\$ INCLUDE(CONBCD)
		451	*****
		452	*
		453	*
		454	CONBCD
		455	*****
		456	=====
		457	*
		458	THIS UTILITY CONVERTS AN 8 BIT BINARY VALUE TO BCD
		459	AT ENTRY:
		460	A = 8 BIT BINARY VALUE
		461	*
		462	AT EXIT:
		463	A = BCD VALUE
		464	*****
0005		465	TEMP1 SET R5
0006		466	TEMP2 SET R6
		467	*
		468	1 CONVERT TO BCD
		469	CONBCD:
		470	1 BCDACC:=0
0137 BE00		471	MOV TEMP2,#00
		472	1 COUNT:=8
0139 BB08		473	MOV COUNT,#08
		474	1 REPEAT
		475	BCDCOR:
		476	2 BIN:=BIN*2
013B 97		477	CLR C
013C F7		478	RLC A
		479	2 BCD:=BCD*2+CARRY
013D AD		480	MOV TEMP1,A
013E FE		481	BCDOC: MOV A,TEMP2
013F 7E		482	ADDC A,TEMP2
0140 57		483	DA A
0141 AE		484	MOV TEMP2,A
0142 FD		485	MOV A,TEMP1
		486	2 IF CARRY FROM BCDACC GOTO ERROR EXIT
0143 F649		487	JC BCDCOD
		488	2 COUNT:=COUNT-1
		489	1 UNTIL COUNT=0
0145 EB3B		490	DJNZ COUNT,BCDCOR
0147 97		491	CLR C ; CLEAR CARRY TO INDICATE NORMAL TERMINATION
0148 FE		492	MOV A,TEMP2 ; PUT BCD VALUE IN ACCUMULATOR FOR RETURN
		493	1 END CONVERT TO BCD
0149 83		494	BCDCOD: RET
		495	\$EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT
		496	\$ INCLUDE(CONB2)
		497	*****
		498	*****
		499	CONBIN
		500	*****
		501	-----
		502	*****
		503	THIS UTILITY CONVERTS A 2 DIGIT BCD VALUE TO BINARY
		504	AT ENTRY:
		505	A = BCD VALUE
		506	*****
		507	AT EXIT:
		508	A = EIGHT BITS OF THE BINARY RESULT
		509	*****
		510	*****
		511	*****
		512	*****
		513	*****
0005		514	TEMP1 SET R5
0006		515	TEMP2 SET R6
0007		516	TEMP3 SET R7
		517	*****
		518	;1 CONVERT_TO_BINARY
		519	CONBIN:
014A BB01		520	;1 COUNT:=DIGITPAIR
		521	MOV COUNT,#DIGPR
		522	; REPEAT
		523	CONBLP:
		524	; BIN:=BIN*10
014C AF		525	MOV TEMP3,A
014D 47		526	SWAP A
014E 530F		527	ANL A,#0FH
0150 345A		528	CALL CONB10
		529	; BIN:=BIN+MEM(RO)[7-4]
0152 AE		530	MOV TEMP2,A
0153 FF		531	MOV A,TEMP3
0154 530F		532	ANL A,#0FH
0156 6E		533	ADD A,TEMP2
		534	; COUNT:=COUNT-1
		535	; UNTIL COUNT=0
0157 EB4C		536	DJNZ COUNT,CONBLP
		537	; END CONVERT_TO_BINARY
0159 83		538	CONBER: RET
		539	*****
		540	\$EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT
		= 541	UTILITY TO MULTIPLY BIN BY 10
		= 542	CARRY WILL BE SET IF OVERFLOW OCCURS
		= 543	
		= 544	
015A	AD	= 545	CONB10: MOV TEMP1,A ; SAVE A
		= 546	
015B	97	= 547	CLR C
015C	F7	= 548	RLC A ; BIN:=BIN*2
		= 549	
015D	F7	= 550	RLC A ; BIN:=BIN*4
		= 551	
015E	6D	= 552	ADD A,TEMP1 ; BIN:=BIN*5
		= 553	
015F	F7	= 554	RLC A ; BIN:=BIN*10
		= 555	
0160	83	= 556	RET
		= 557	
		= 558	
		= 559	\$EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT
		560	END

USER SYMBOLS

ANGLE 0023	BAK1 00AB	BAK2 00B5	BCDCOB 013B	BCDCOD 0149
BCDOC 013E	BCDTR 0021	BINTR 0020	CLRFLG 004C	CNBCD 0137
CNTDWN 00C1	CONB10 015A	CONBER 0159	CONBIN 014A	CONBLP 014C
CONT2 0111	CONT3 0124	CONTON 0058	COUNT 0003	DECREM 0049
DELTA 0026	DIG1 0123	DIG2 011F	DIGPR 0001	DISFLG 0025
DLGT10 0007	FIRE 00C5	HERE1 0137	HOLD 0054	ICNT 0004
INCREM 003B	INIT 0000	INITZ 0098	INWAIT 0063	IPWAIT 0016
LDA 0000	LOOK 004F	NOSWAP 010E	OFFTIM 0003	OTCALC 0038
PASTOR 0022	PHLTSV 0006	PHSANG 0005	PWRON 0002	REFRSH 0100
SEG DAT 0024	SELRBO 00B1	SELRB1 00A7	SETVAL 0004	STA 0001
START 000E	TAB 00C9	TABL 012D	TEMP1 0005	TEMP2 0006
TEMP3 0007	TIME 0002	TRIAC 00BB	VOWAIT 0006	WAITST 000A
XA 0002				

ASSEMBLY COMPLETE, NO ERRORS

[illegible]

CROSS REFERENCE COMPLETE

5V EPROM and ROMs

INTRODUCTION	4-2
PINOUT EVOLUTION	4-2
SYSTEM ARCHITECTURE	4-2
BUS CONTENTION	4-3
THE MICROPROCESSOR/MEMORY INTERFACE	4-4
TERMINOLOGY	4-4
THE NEW INTEL FAMILY	4-5
PIN DEVICES AND PIN SITES	4-6
PRINTED CIRCUIT BOARD DESIGN	4-7

INTRODUCTION

This Application Note discusses how the new Intel family of 5 volt EPROMs and ROMs can be used with microprocessor systems. The pinout evolution and philosophy are explored in detail, which leads directly to system architecture. Particular emphasis will be placed on the pitfalls of bus contention and the microprocessor/memory interface. Finally, an actual printed circuit board layout is presented.

PINOUT EVOLUTION

As EPROM/ROM technology has evolved, there are often periods of confusion over EPROM and ROM pinouts, as ROM density usually leads EPROM density by a factor of two, but ultimately users want any given EPROM to have a ROM compatible part. As we have seen, after the 2716 16K EPROM was introduced, a new ROM pinout emerged and "triumphed" over an earlier "standard." The reason this ROM pinout change occurred is that as codes stabilize in user's systems and equipment, many users opt for the less expensive ROMs, which are mask programmable devices. At the same time, users often use the highest available density ROM so they combine modular firmware and minimize device count. Of course, many users never do go to the ROM stage with their equipment, preferring to minimize inventory levels and utilize standard designs that can be customized for final equipment configurations, but they always want the capability to do so if desired.

In addition, over the past few years, the development of microprocessors has been intimately entwined with both ROMs and EPROMs.

The 1702A and its ROM counterpart, the 1302, were completely adequate to support the requirements of the 4004 series of microprocessors. In order to support the 5 volt, 3MHz 8085A and 5MHz 8086, it is desirable to use a compatible device such as the Intel 5 volt 2716, whose 450ns access time is compatible with the microprocessor requirements. Some high performance versions of these processors may require selected versions of the 2716 (such as the 2716-1 with $t_{ACC}=350ns$, or the 2716-2 with $t_{ACC}=390ns$) depending on the actual system configuration.

Summarizing these events since the introduction of the Intel 1702A, which was the first EPROM, we can postulate the following hypothesis: at any point in time, the present EPROM determines the pinout for the next generation ROM. And, if the subsequent larger density EPROM is not ROM compatible, the ROM will change. Also, it can be seen that ROMs and EPROMs must evolve along with microprocessor developments—so memory performance does not limit system performance.

The devices which are discussed in this Application Note represent an extension of the 5 volt compatible family to 32K bit and 64K bit densities, while improving performance as discussed above. It also follows that the pinout

for the 32K devices must be derived from the 2716 in order to maintain socket compatibility. This 16K to 32K pinout evolution is shown in Figure 1.

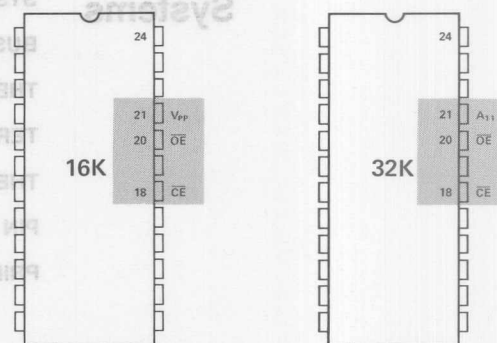


Figure 1. 16K EPROM Determines 32K ROM Pinout

SYSTEM ARCHITECTURE

As higher performance microprocessors have become available, the architecture of microprocessor systems has been evolving, again placing demands on memory. For many years, system designers have been plagued with the problem of bus contention when connecting multiple memories to a common data bus. There have been various schemes for avoiding the problem, but device manufacturers have been unable to design internal circuits that would guarantee that one memory device would be "off" the bus before another device was selected. With small memories (512x8 and 1Kx8), it has been traditional to connect all the system address lines together and utilize the difference between t_{ACC} and t_{CO} to perform a decode to select the correct device (as shown in Figure 2).

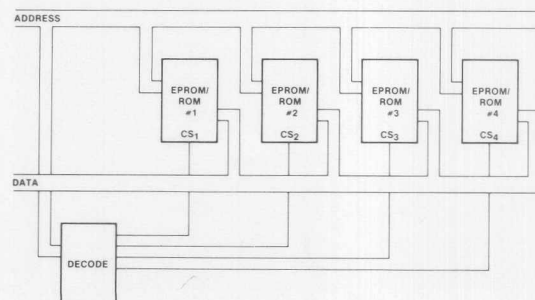


Figure 2. Single Control Line Architecture

With the 1702A, the chip select to output delay was only 100ns shorter than the address access time; or to state it another way, the t_{ACC} time was 1000ns while the t_{CO} time was 900ns. The 1702A t_{ACC} performance of 1000ns was suitable for the 4004 series microprocessors, but the 8080 processor required that the corresponding numbers be reduced to $t_{ACC} = 450$ ns and $t_{CO} = 120$ ns. This allowed a substantial improvement in performance over the 4004 series of microprocessors, but placed a substantial burden on the memory. The 2708 was developed to be compatible with the 8080 both in access time and power supply requirements. A portion of each 8080 machine cycle time had to be devoted to the architecture of the system decoding scheme used. This devoted portion of the machine cycle included the time required for the system controller (8224) to perform its function before the actual decode process could begin.

Let's pause here and examine the actual decode scheme that was used so we can understand how the control functions that a memory device requires are related to system architecture.

The 2708 can be used to illustrate the problem of having a single control line. The 2708 has only one read control function, chip select (\overline{CS}), which is very fast ($t_{CO} = 120$ ns) with respect to the overall access time ($t_{ACC} = 450$ ns) of the 2708. It is this time difference (330ns) that is used to perform the decode function, as illustrated in Figure 3. The scheme works well and does not limit system performance, but it does lead to the possibility of bus contention.

BUS CONTENTION

There are actually two problems with the scheme described in the previous section. First, if one device in a multiple memory system has a relatively long deselect time, and a relatively fast decoder is used, it would be possible to have another device selected at the same time. If the two devices thus selected were reading opposite data; that is, device number one reading a HIGH and device number two reading a LOW, the output transistors of the two memory devices would effectively produce a short circuit, as Figure 4 illustrates. In this case, the current path is from V_{CC} on device number one to GND on device number two. This current is limited only by the "on" impedance of the MOS output transistors and can reach levels in excess of 200mA per device. If the MOS transistors have a lot of "extra" margin, the current is usually not destructive; however, an instantaneous load of 400mA can produce "glitches" on the V_{CC} supply — glitches large enough to cause standard TTL devices to drop bits or otherwise malfunction, thus causing incorrect address decode or generation.

The second problem with a single control line scheme is more subtle. As previously mentioned, there is only one control function available on the 2708 and any decoding scheme must use it out of necessity. In addition, any inadvertent changes in the state of the high order address lines that are inputs to the decoder will cause a change in

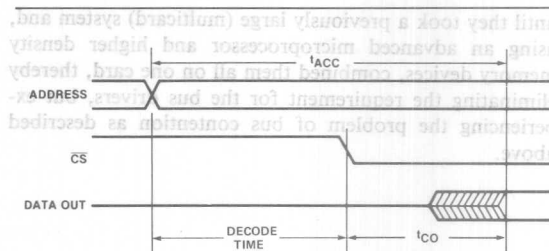


Figure 3. Single Line Control Architecture

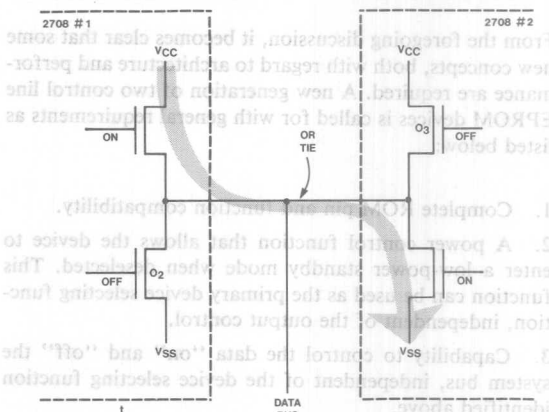


Figure 4. Results of Improper Timing when OR Tying Multiple Memories

the device that is selected. The result is the same as before — bus contention, only from a different source. The deselected device cannot get "off" the bus before the selected one is "on" the bus as the addresses rapidly change state. One approach to solving this problem would be to design (and specify as a maximum) devices with t_{DF} time less than t_{CO} time, thereby assuring that if one device is selected while another is simultaneously being deselected, there would be some small (20ns) margin. Even with this solution, the user would not be protected from devices which have very fast t_{CO} times (t_{CO} is specified as a maximum).

The only sure solution appears to be the use of an external bus driver/transceiver that has an independent enable function. Then that function, not the "device selecting function," or addresses, could control the flow of data "on" and "off" the bus, and any contention problems would be confined to a particular card or area of a large card. In fact, many systems are implemented that way — the use of bus drivers is not at all uncommon in large systems where the drive requirements of long, highly capacitive interconnecting lines must be taken into consideration — it also may be the reason why more system designers were not aware of the bus contention problem

preventing the problem of bus contention as described above.

THE MICROPROCESSOR/MEMORY INTERFACE

From the foregoing discussion, it becomes clear that some new concepts, both with regard to architecture and performance are required. A new generation of two control line EPROM devices is called for with general requirements as listed below:

1. Complete ROM pin and function compatibility.
2. A power control function that allows the device to enter a low-power standby mode when deselected. This function can be used as the primary device selecting function, independent of the output control.
3. Capability to control the data "on" and "off" the system bus, independent of the device selecting function identified above.
4. Access time compatible with the high performance microprocessors that are currently available.

Now let's examine the system architecture that is required to implement the two line control and prevent bus contention. This is shown in the form of a timing diagram (Figure 5). As before, addresses are used to generate the unique device selecting function, but a separate and independent Output Enable (OE) control is now used to gate data "on" and "off" the system data bus. With this scheme, bus contention is completely eliminated as the processor determines the time during which data must be present on the bus and then releases the bus by way of the Output Enable line, thus freeing the bus for use by other devices, either memories or peripheral devices. This type of architecture can be easily accomplished if the memory devices have two control functions, and the system is implemented according to the block diagram shown in Figure 6. It differs from the previous block diagram (shown in Figure 2) in that the control bus, which is connected to all memory Output Enable pins, provides separate and independent control over the data bus. In this way, the microprocessor is always in control of the system; while in the previous system, the microprocessor passed control to the particular memory device and then waited for data to become available. Another way to look at it is, with a single control line the system is always asynchronous with respect to microprocessor/memory communications. By using two control lines, the memory is synchronized to the processor.

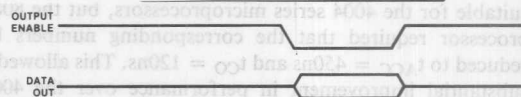


Figure 5. Two Control Line Architecture

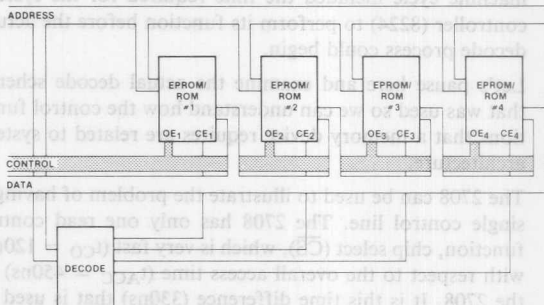


Figure 6. Two Control Line Architecture

TERMINOLOGY

Some of the terminology applied to the functions of the Intel 5 volt compatible family may be confusing or unfamiliar to many EPROM/ROM users, so the various terms are defined here. Actually, the nomenclature was developed by various standards groups and is reiterated here to avoid confusion as we begin a detailed discussion of the devices themselves.

First of all, Chip Enable (CE) must be defined, as it is the primary device selection pin. By agreed standards, that function which substantially affects power dissipation is called CE. Any memory device that has a CE function has both an active and standby power level associated with it.

Output Enable (OE) is the signal that controls the output. The fundamental purpose of OE is to provide a completely separate means of controlling the output buffer of the memory device, thereby eliminating bus contention.

Chip Select (CS) is a signal that gets logically ANDed with addresses. In a completely static device, CS must remain stable throughout the entire device cycle, and its function is equivalent to Output Enable (OE).

THE NEW INTEL FAMILY

Figure 7 shows the new Intel 5 volt compatible family of EPROMs and ROMs. In order to take advantage of the modular compatibility offered by the family, the functional compatibility of device pins 18, 19 and 21 must be understood. (Shaded area in Figure 7.)

First, we must examine the compatibility of the two oldest EPROM members of the 5 volt family — the 8K (2758) and the 16K (2716).

Pin 21 (V_{PP}) is normally connected to V_{CC} for read only applications of both devices, and pin 19 is either at GND (V_{IL}) for the 8K 2758 or connected to A_{10} for the 16K 2716. Further details on either of these devices can be found in Section 9 of the 1977 Edition of the Intel Memory Design Handbook, or Section 4 of the 1978 Intel Data Catalog.

The 32K (4Kx8) devices, which have identical pinouts for both the ROM and EPROM, will now be discussed. Pin 18 is \overline{CE} . Pin 19 is A_{10} , while pin 20 is \overline{OE} . As was pointed out before, Output Enable is the function which allows independent control of the data "on" and "off" the output bus. As Figure 7 indicates, V_{PP} (the programming voltage for the 2732 EPROM) is now multiplexed with \overline{OE} on pin 20. Pin 21 becomes A_{11} , which is the additional address bit that is required as the density increases from 16K to 32K.

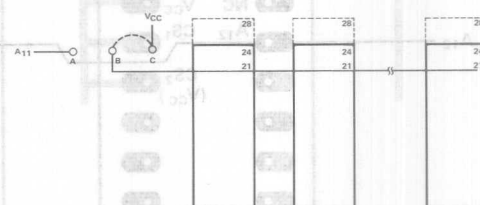
Pin 21 is the only pin that requires any special consideration when designing a system to accept the 8K, the 16K, or the 32K device. With the 8K and the 16K devices, pin 21 must be connected to V_{CC} , while with the 32K and higher density devices, it must be connected to A_{11} . This is easily accomplished by making sure the printed circuit trace links all pin 21's together as though they were an address line and allowing for a jumper that will connect pin 21 to either V_{CC} or A_{11} at the edge of the array (this technique can be seen in the "Printed Circuit Board Design" section and in Figure 8). Connecting the pin 21's together in this manner is acceptable as the read current requirement for V_{PP} is 4mA maximum per device — low enough to be handled by a signal trace, but too high for an address driver to provide directly.

The highest density member of the family is a 64K ROM which is also shown in Figure 7. In order to maintain total compatibility it is packaged in a standard 28-pin package.

It may seem as though the 28 pin package is not compatible with the rest of the family, but referring again to Figure 7, note that the lower 24 pins are identical to the 24 pin 8K, 16K and 32K devices. To allow for total compatibility within the family: printed circuit boards must be laid out to accommodate 28 pin sites; a jumper must be included to accommodate pin 21 as shown in Figure 8, and when using 64K devices, CS_2 (Pin 26) must be mask coded active high. This compatibility can also be seen graphically in Figures 9 and 10. The upper portion of the figure shows how 24 pin devices are used in the 28 pin sites. The two control lines (\overline{CE} and \overline{OE}) remain unchanged as discussed earlier, and A_{12} , the next address bit required for a 64K bit device, is connected to pin 2 of the 28 pin site. The lower portion of the figure illustrates the use of 28 pin devices. Address bit A_{12} is already connected to the right pin, and the chip selects (CS_1 and CS_2) are connected to the V_{CC} power distribution grid. This configuration would require that both CS_1 and CS_2 be coded active high.



32K Bit Density and Higher



8K and 16K Density Devices

Figure 8. Pin 21 Connections for Various Density Devices

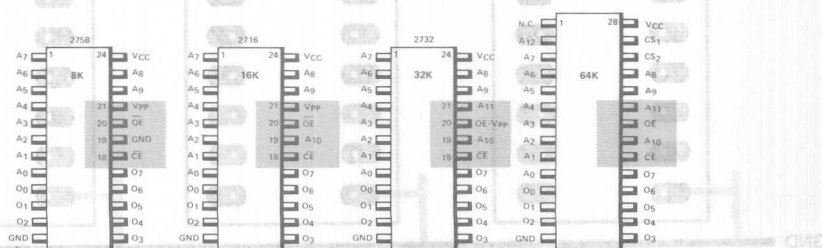


Figure 7. 5 Volt EPROM/ROM Compatible Family

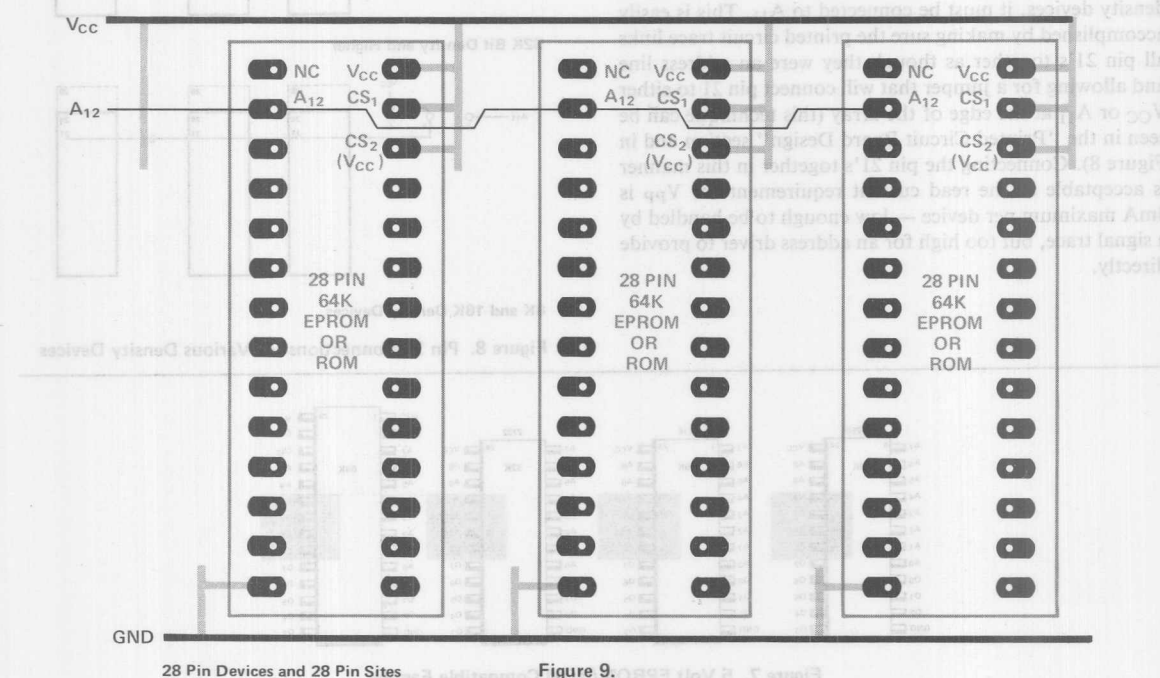
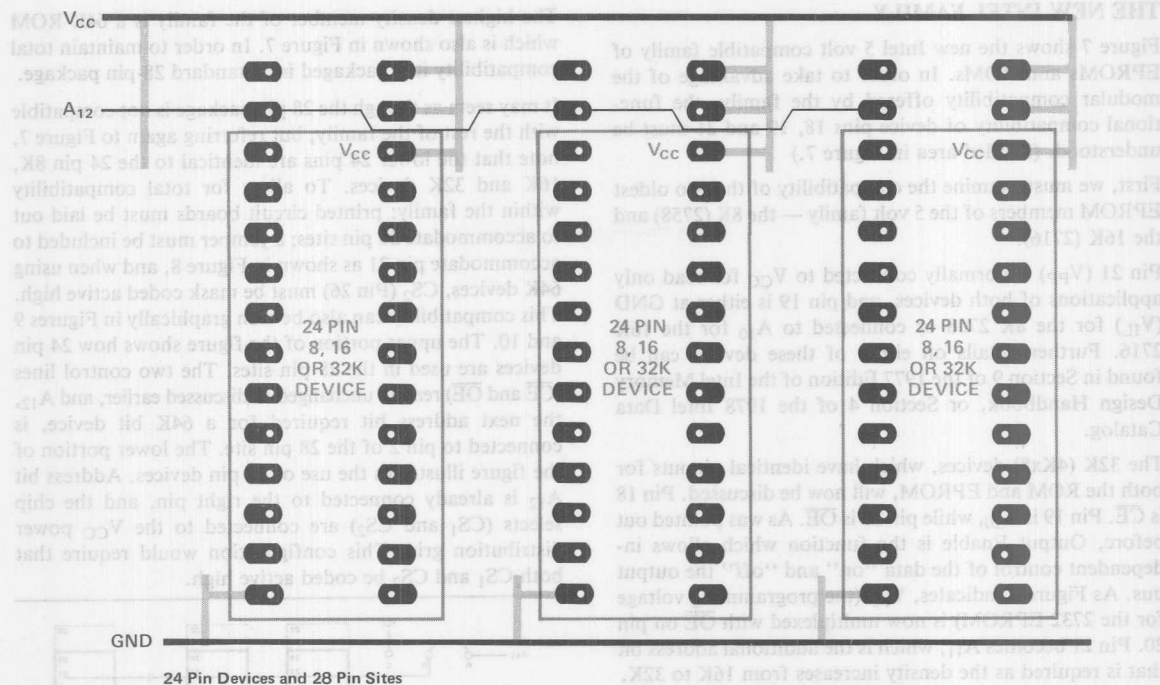


Figure 9.

In some systems, additional logic may be used to implement a "special" decode of CS_1 to allow for ROM to overlap RAM when a system bootstrap program is being loaded; that additional logic should be implemented with CS_1 ; CS_2 should be coded active high in order to preserve total compatibility.

To summarize, the selection of a 28 pin package for 64K devices has several benefits of importance to present and future system designs:

1. Two line control philosophy (separate \overline{CE} and \overline{OE} functions) is preserved at the 64K bit level.
2. 64K EPROM compatibility is allowed for by maintaining a pin for the V_{PP} function.
3. The next generation (128K bit ROM) must be in a 28 pin package.

If CS_2 (pin 26) is mask coded to be active high and connected to V_{CC} , and the jumper provision for pin 21 is included on the card as described above, any member of the family can be plugged into the same socket — 1K, 2K, 4K or 8K bytes — without any card modification or redesign. In addition, future devices of higher density will fit in the same pinout.

PRINTED CIRCUIT BOARD DESIGN

The I_{CC} waveform for the 2332 and the 2364 is shown in Figure 10. The supply current, I_{CC} , has three segments that are of concern to the system designer — the standby level, active level and the transient peaks that are produced on the rising and falling edges of Chip Enable. The transient currents must be suppressed by properly selected decoupling capacitors. High quality, high frequency ceramic capacitors of small physical size with low inherent inductance should be used. In addition, bulk decoupling must be provided, usually near where the power supply is connected to the array. The purpose of the bulk decoupling is to overcome the voltage droop caused by the inductive effects of the PC board traces. Electrolytic or tantalum capacitors are suitable for bulk decoupling. The following capacitance values and locations are recommended for the 2332 and 2364:

1. A $0.1\mu F$ ceramic capacitor between V_{CC} and GND at every other device.
2. A $4.7\mu F$ electrolytic capacitor between V_{CC} and GND for each eight devices.

A printed circuit board layout for a total array of 16 devices is shown in Figure 11. This printed circuit layout incorporates a power supply distribution system such that the power supply and ground traces on the PC board are

gridded both vertically and horizontally at each memory device; this technique minimizes the power distribution system impedance and enhances the effect of the decoupling capacitors. Provisions are included for all address inputs, output enable inputs, data outputs and decoded chip enable inputs. The $0.1\mu F$ capacitors referred to above are included for every other device (indicated by the legend C2) while the bulk decoupling capacitor is shown at the upper left-hand corner (indicated by the legend C1). The layout consists of four rows of four 28-pin device sites each and embodies all of the concepts explained above. Note that pins 28, 27 and 26 are all connected to V_{CC} . This requires that when ordering mask programmed 2364 64K ROMs, the order must specify that CS_1 and CS_2 be coded active HIGH. The single jumper provision discussed in the previous section is also included at the upper left-hand corner of the array (indicated by A, B, and C). Pad B is connected to pin 21 of all devices in the array; pad A should be connected to the A_{11} address driver and pad C is connected to V_{CC} . For use with 32K bit or larger devices, a jumper must be installed between pads A and B; for use with the 2716 (16K) or the 2758 (8K), the jumper must be installed between pads B and C.

A full size (2x) artwork film is included on the last page of this Application Note. The entire array, or segments of it can be photographed and used directly as part of a system board.

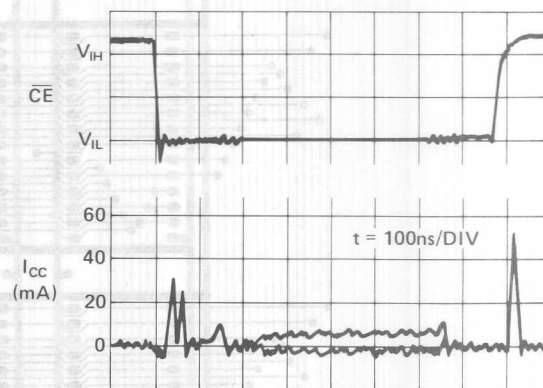


Figure 10. Typical I_{CC} Current vs Time

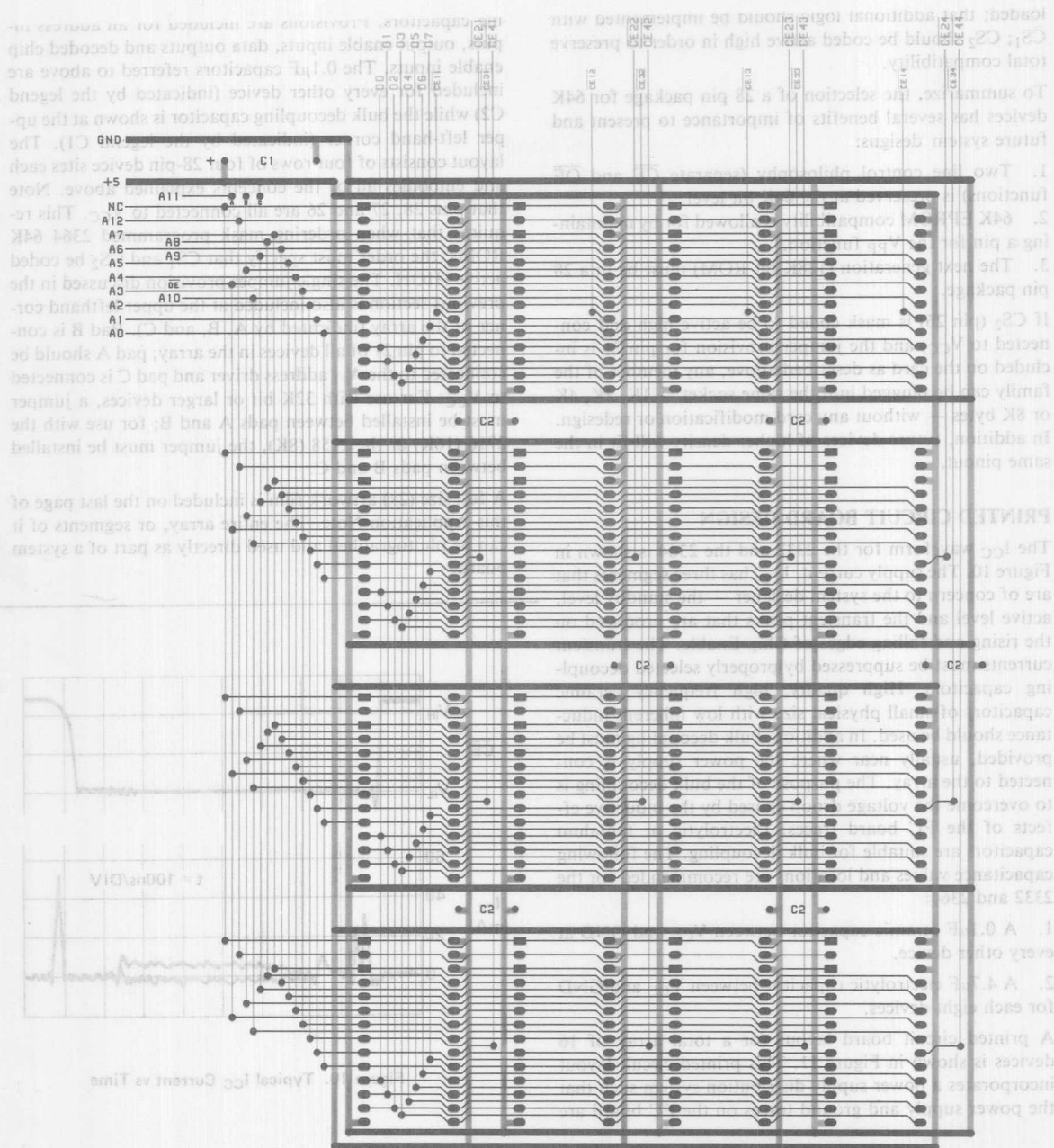


Figure 11. Printed Circuit Board Layout of 16 Devices

CRYSTALS: Specifications for Intel Components

Contents

INTRODUCTION	4-10
CRYSTAL OPERATION — BRIEF THEORETICAL EXPLANATION	4-10
CIRCUIT CONFIGURATIONS FROM VARIOUS MANUALS/EXPLANATION	4-11
Series 10 pF Capacitor	4-11
Parallel 20 pF Capacitor to Ground	4-11
8224 Overtone Application (Tank Circuit)	4-11
Precise Timing Applications	4-12
WHAT IF I USE A CRYSTAL OTHER THAN SPECIFIED?	4-12
SPECIFICATIONS	4-12
Intel Component Crystal Requirements	4-12
Suggested Suppliers, Part Numbers	4-13

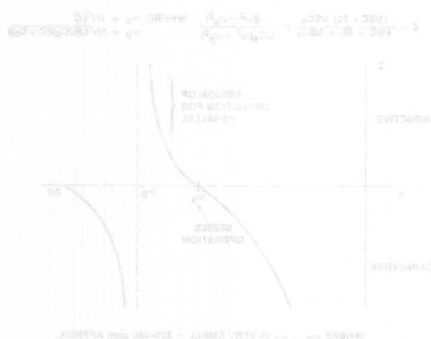


Figure 1

When operating at series resonance (ω_s), the equivalent circuit of the crystal becomes a simple resistor R_s (Figure 3). Impedance R_s was neglected in the impedance calculation. This R_s value must be specified to the crystal vendor when buying a crystal.

This parameter becomes a problem with lower frequency or overtone crystals (thicker, more resistance) and a buffer that doesn't have sufficient gain to drive those crystals (i.e., loop gain becomes less than 1). Overtone crystals also have R_s problems as their R_s is associated with the fundamental frequency of the crystal, not the 3rd harmonic or overtone. The 8224 is particularly sensitive to R_s with 32 MHz overtone applications.



Figure 2

When giving the specifications to a crystal vendor for a crystal, it is helpful to understand its equivalent circuit, as shown in Figure 1. The impedance of this circuit (neglecting R_s to simplify matters for conceptual purposes) can be calculated and plotted against frequency (Figure 2). This frequency-impedance plot illustrates the two different operating modes of crystal, ω_s (series resonance) occurs when the impedance (resistance) is zero and ω_p (parallel resonance) occurs when the impedance goes to infinity and appears inductive.

Different LSI components prefer different crystal due to the nature of their internal oscillator design. In general, Intel bipolar components have a non-inverting, bidirectional drive oscillator, whereas CMOS components use an inverting oscillator. Non-inverting oscillators prefer series resonant crystals (as the series resonant crystal has 0 degree net phase shift), while inverting oscillators prefer parallel resonant crystals which are parallel resonant. Since a crystal has both a series and parallel operating frequency, many times any crystal will seem to work when connected to a component.

Crystals are piezoelectric devices which transform voltage energy to mechanical vibrations and voltage oscillations. The frequency of the crystal is largely dependent on its thickness, with thinner crystals producing a higher frequency.

Crystals are generally specified as being series or parallel resonant, but all crystals are in actually both. Vendors supply crystals as series or parallel resonant based on the desired frequency and the crystal's relative ability to generate the frequency in that mode. On a conceptual basis, when using a crystal as series resonant, its output is in phase with its input, whereas using the crystal as parallel resonant will result in a phase shift from its input to output.

CRYSTAL OPERATION — BRIEF THEORETICAL EXPLANATION

Understanding Crystal Operation

Crystals are piezoelectric devices which transform voltage energy to mechanical vibrations and voltage oscillations. The frequency of the crystal is largely dependent on its thickness, with thinner crystals producing a higher frequency.

Crystals are generally specified as being series or parallel resonant, but all crystals are in actually both. Vendors supply crystals as series or parallel resonant based on the desired frequency and the crystal's relative ability to generate the frequency in that mode. On a conceptual basis, when using a crystal as series resonant, its output is in phase with its input, whereas using the crystal as parallel resonant will result in a phase shift from its input to output.

Different LSI components prefer different crystal due to the nature of their internal oscillator design. In general, Intel bipolar components have a non-inverting, bidirectional drive oscillator, whereas CMOS components use an inverting oscillator. Non-inverting oscillators prefer series resonant crystals (as the series resonant crystal has 0 degree net phase shift), while inverting oscillators prefer parallel resonant crystals which are parallel resonant. Since a crystal has both a series and parallel operating frequency, many times any crystal will seem to work when connected to a component.

When giving the specifications to a crystal vendor for a crystal, it is helpful to understand its equivalent circuit, as shown in Figure 1. The impedance of this circuit (neglecting R_s to simplify matters for conceptual purposes) can be calculated and plotted against frequency (Figure 2). This frequency-impedance plot illustrates the two different operating modes of crystal, ω_s (series resonance) occurs when the impedance (resistance) is zero and ω_p (parallel resonance) occurs when the impedance goes to infinity and appears inductive.

INTRODUCTION

The following brief note is intended to answer the simpler questions on crystal specifications and their operation with the various Intel components. First, a theoretical explanation of the crystal is given to aid the user in understanding crystal operation. This includes a discussion of the parameters necessary for proper specification to the vendor. Following this section are explanations of the various crystal-capacitor configurations seen in the Intel User's Manuals and data sheets; why they are suggested for proper crystal operation and what might happen if they weren't there.

The final section of this note provides a list of suggested crystal specifications, suppliers, and part numbers for the highest frequency crystals possible for the various Intel components that require them. In no way does this list represent the only crystals or suppliers available. This section is conveniently preceded by a discussion of problem areas that may result if a user is using the wrong crystal required for the component.

CRYSTAL OPERATION — BRIEF THEORETICAL EXPLANATION

Understanding Crystal Operation

Crystals are piezoelectric devices which transform voltage energy to mechanical vibrations and voltage oscillations. The frequency of the crystal is largely dependent on its thickness, with thinner crystals producing a higher frequency.

Crystals are generally specified as being series or parallel resonant, but all crystals are in actuality both. Vendors supply crystals as series or parallel resonant based on the desired frequency and the crystal's relative ability to generate the frequency in that mode. On a conceptual basis, when using a crystal as series resonant, its output is in phase with its input, whereas using the crystal as parallel resonant will result in a phase shift from its input to output.

Different LSI components prefer different crystals due to the nature of their internal oscillator design. In general, Intel bipolar components have a non-inverting, bidirectional drive oscillator, whereas NMOS components use an inverting oscillator. Non-inverting oscillators prefer series resonant crystals (as the series resonant crystal has 0 degree net phase shift), while inverting oscillators prefer crystals which are parallel resonant. Since a crystal has both a series and parallel operating frequency, many times any crystal will seem to work when connected to a component.

When giving the specifications to a crystal vendor for a crystal, it is helpful to understand its equivalent circuit as shown in Figure 1. The impedance of this circuit (neglecting R to simplify matters for conceptual purposes) can be calculated and plotted against frequency (Figure 2). This frequency-impedance plot illustrates the two different operating modes of crystal. ω_s (series resonance) occurs when the impedance (reactance) is zero and ω_p (parallel resonance) occurs when the impedance goes to infinity and appears inductive.

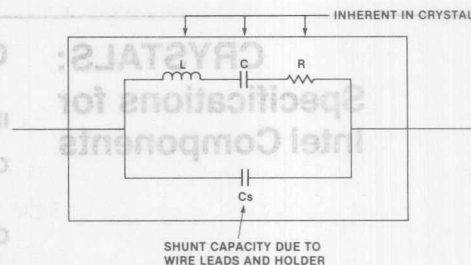


Figure 1

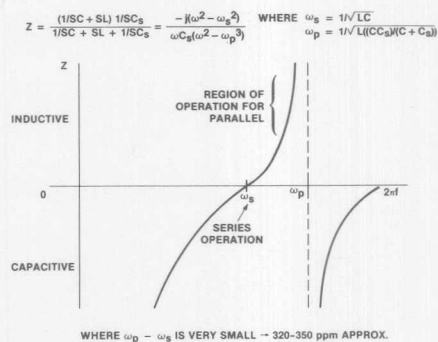


Figure 2

When operating at series resonance (ω_s) the equivalent circuit of the crystal becomes a simple resistor R_s (Figure 3; remember, R was neglected in the impedance calculation). This R_s value must be specified to the crystal vendor when buying a crystal.

This parameter becomes a problem with lower frequency or overtone crystals (thicker, more resistance) and a buffer that doesn't have sufficient gain to drive those crystals (i.e., loop gain becomes less than 1). Overtone crystals also have R_s problems as their R_s is associated with the fundamental frequency of the crystal, not the 3rd harmonic or overtone. The 8224 is particularly sensitive to R_s with 27 MHz overtone applications.



Figure 3

Conversely, if operating at ω_p (parallel resonance), the crystal appears inductive in the circuit (Figure 4). Since the crystal appears inductive, any changes in reactance that the crystal sees will have the effect of pulling the frequency of the crystal. As a result of this, the amount of load capacitance seen by the crystal in the circuit configuration becomes important. This load capacitance, CL, is the dynamic capacity of the total circuit measured across the terminals of the crystal. The amount of this capacitance should always be specified to the crystal vendor if the crystal will be operating at parallel resonance.

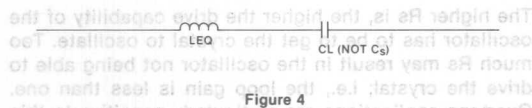


Figure 4

CIRCUIT CONFIGURATIONS FROM VARIOUS MANUALS/EXPLANATIONS

Series 10 pF Capacitor Included (Figure 5)

This additional capacitor is recommended at times to debias the crystal. Due to the component's internal circuit, a small DC bias may exist across the crystal which would strain the crystalline structure. It is also provided for trimming the frequency of the crystal to compensate for the loading effects of the component.

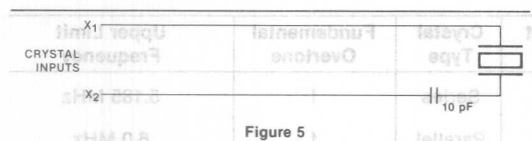


Figure 5

Parallel 20 pF Capacitors to Ground (Figure 6)

Crystals can oscillate at several different frequencies, each emanating from a different direction of vibration in the crystal. For a crystal to oscillate during startup in its fundamental frequency, it is best for the crystal to see the slew rate (Figure 7) of the pulse provided from the oscillator to be as close to the operating frequency as possible.

These 20 pF capacitors act as a high frequency filter to create a slew rate closer to the fundamental frequency of the crystal. As can be guessed, lower frequency crystals are more susceptible to the problem of not starting up in the fundamental frequency.

Capacitors are placed on both sides of the crystal as some components have bidirectional drive buffers (i.e., 1/2 of cycle drive from one side, other half from opposite side). A crystal that needs these extra 20 pFs to ground will be characterized by starting up at a 3rd or 5th harmonic instead of the fundamental frequency. The CL specifications in the specification section takes into consideration these extra 20 pF capacitors required for some Intel components for proper operation.

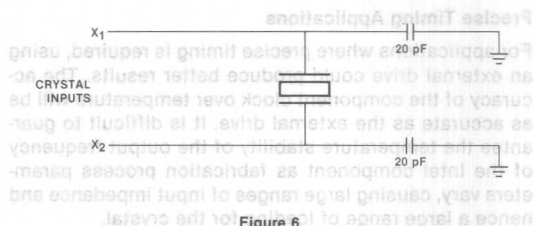


Figure 6



Figure 7

Tank Circuit

On some Intel components, provision is made for a tank circuit. This is for the use of an overtone crystal; i.e., one that is working at a harmonic (generally its 3rd). The tank circuitry is a filter to bypass the lower and higher, unwanted frequencies to ground while appearing "open" to the desired frequency. It is necessary to use tank circuits and overtone crystals when in the 25+ MHz range and above. Fundamental crystals are difficult to make in this frequency range as the crystal must be thinner for higher frequencies.

A circuit that has been used for the 8224 in 27 MHz overtone crystal applications is shown in Figure 8.

This filter can be approximated through formulas where afterwards it will be necessary to tweak the component values for optimization. The formula used to get the original component values is:

$$f = \frac{1}{2\pi\sqrt{L_1 C_1}} \quad \text{where } f = \text{overtone frequency}$$

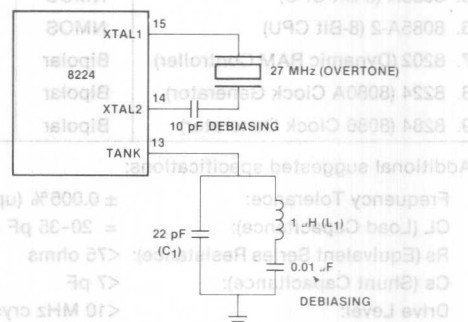


Figure 8

as accurate as the external drive. It is difficult to guarantee the temperature stability of the output frequency of the Intel component as fabrication process parameters vary, causing large ranges of input impedance and hence a large range of loading for the crystal.

WHAT IF I USE A CRYSTAL OTHER THAN SPECIFIED?

Series vs. Parallel

As discussed in the theoretical section, all crystals have a series and parallel operating mode. Placing a series crystal on a device requiring a parallel (i.e., there is an inverting oscillator between the two inputs) will force the crystal to oscillate in its parallel mode (and vice versa). A system with the wrong crystal will exhibit its clock frequency shifted a small percentage (about 320 to 350 parts per million) from the specified crystal frequency. When using the wrong crystal, any attempts to trim the frequency to its specified value by using small parallel (series if series crystal) variable capacitors will cause the crystal to stop oscillating, as predicted by theory. If the correct crystal is being used, trimming can be done. In applications where accuracy is not important, series crystals are sometimes substituted for parallel in the circuit. For instance, the 8048 has been characterized to be compatible with the series color burst TV crystal

Insufficient Drive Level

The drive level specified is the maximum amount of power that is expected for the crystal to dissipate. If the crystal can't handle this level, frequency drift may occur or possible fracture of the crystal. In other words, if the crystal used cannot handle the oscillator drive level, long term reliability problems may occur.

Rs Too High

The higher Rs is, the higher the drive capability of the oscillator has to be to get the crystal to oscillate. Too much Rs may result in the oscillator not being able to drive the crystal; i.e., the loop gain is less than one. Overtone applications are particularly sensitive to this as thicker crystals are used (lower fundamental frequency, more resistance).

SPECIFICATIONS

Intel Component Crystal Requirements

The following is a list of suggested specifications for crystals to be used with Intel components. In most instances the upper frequency limit is given, with exceptions being footnoted.

Component (Function)	Process	Component Divide By	Crystal Type	Fundamental Overtone	Upper Limit Frequency
1. 4201A (Clock Generator)	CMOS	—	Series	f	5.185 MHz
2. 8035/48/49, 8748 (8-Bit CPU)	NMOS	15	Parallel	f	6.0 MHz
3. 8748/8035-8 (8-Bit CPU)	NMOS	15	Parallel	f	3.6 MHz
4. 8041/8741 (Universal Peripheral Interface)	NMOS	15	Parallel	f	6.0 MHz
5. 8085A (8-Bit CPU)	NMOS	2	Parallel	f	6.25 MHz/6.144 MHz ⁽¹⁾
6. 8085A-2 (8-Bit CPU)	NMOS	2	Parallel	f	10.0 MHz
7. 8202 (Dynamic RAM Controller)	Bipolar	—	Series	f	25 MHz
8. 8224 (8080A Clock Generator)	Bipolar	—	Series	f/o	27 MHz/18.432 MHz ⁽²⁾
9. 8284 (8086 Clock Generator)	Bipolar	3	Series	f	24 MHz/15 MHz ⁽³⁾

Additional suggested specifications:

Frequency Tolerance:	± 0.005% (up to the user)
CL (Load Capacitance):	= 20-35 pF (not necessary when specifying series)
Rs (Equivalent Series Resistance):	<75 ohms
Cs (Shunt Capacitance):	<7 pF
Drive Level:	<10 MHz crystal 10 milliwatts >10 MHz crystal 5 milliwatts

- Notes:**
- 6.144 MHz is commonly used as convenient baud rates can be generated from this frequency.
 - 27 MHz is max. 18.432 is common crystal used which gives maximum clock rate for 8080A. Fundamental crystal should be used for the 18.432 MHz application.
 - Used for either a 8 or 5 MHz output clock, respectively.

Holder specifications are up to the user. A standard popular one that provides ample lead length is HC-33/U (0.750"W x 0.765"H, 1.5" lead length with spacing of 0.486") and can be used for frequencies up to 4 MHz. After 4 MHz a smaller holder can be used such as HC-18/U (0.435"W x 0.530"H, 1.5" lead length with spacing of 0.192"). All crystals listed in the following table will fit in the HC-33/U holder. Other standard holders are available.

Suggested Suppliers, Part Numbers

The following are two vendors (which are among many) that supply crystals to the specifications given earlier and their part numbers (given in order of frequency). The user should make sure that the holder type associated with these part numbers is acceptable in their application.

f	Parallel/ Series	Crystek ⁽¹⁾ Corp.	CTS Knight, ⁽²⁾ Inc.
3.6 MHz	P	**	**
5.185 MHz	S	CY8A	**
6.0 MHz	P	**	MP060
6.144 MHz	P	**	MP061
6.25 MHz	P	**	MP062
10.0 MHz	P	**	MP10A
15.0 MHz	S	CY15A	MP150
18.432	S	CY19B*	MP184*
24.0 MHz	S	**	MP240
25.0 MHz	S	**	MP250
27.0 MHz	S (overtone)	CY27A	MP270

*Intel also supplies a crystal numbered 8801 for this application.

**Contact vendor with the appropriate specifications.

Notes: 1. Address: 1000 Crystal Drive, Fort Meyers, Florida 33901
2. Address: 400 Reimann Ave., Sandwich, Illinois

The user is not limited to these vendors or frequencies. The frequency chosen by the user should take into consideration convertibility to desired baud rates and the system timings that must be met.

In summary, to obtain a crystal for the user's application, it is necessary to give the crystal vendor the following information:

Series or parallel
Fundamental or overtone
Rs (series), Cs (shunt)
CL if parallel
Drive Level
Frequency tolerance
Holder type

For a select few crystals, vendor numbers were given for two different vendors. With the above information, most vendors can make the desired crystal whether or not they have it as a standard part.

